Advection:
An Introduction

Lloyd Fosdick

September 5, 1995

High Performance Scientific Computing
University of Colorado at Boulder
Copyright ©1995 by the HPSC Group of the University of Colorado

The following are members of
the HPSC Group of the Department of Computer Science
at the University of Colorado at Boulder:

   Lloyd D. Fosdick
   Elizabeth R. Jessup
   Carolyn J. C. Schauble
   Gitta O. Domik
Advection

Contents

1 Introduction 1

2 One-dimensional advection equation 4
2.1 Advection equation is \( d\phi/dt = 0 \) .......................... 6
2.2 Advection equation as a conservation law .................. 6
2.3 Boundary conditions ........................................... 7

3 Numerical methods: first order 12
3.1 Discretization of \( R \) ............................................. 12
3.2 First-order difference methods ............................... 13
3.2.1 A forward difference algorithm, Algorithm 1 ............. 13
3.2.2 Test of the forward-difference algorithm, Algorithm 1 .. 15
3.2.3 A backward difference algorithm, Algorithm 2 ........... 16
3.2.4 Test of the backward difference algorithm, Algorithm 2 18
3.2.5 Analysis of Algorithms 1 and 2 .......................... 18
3.3 Confirmation of analysis of Algorithm 2 ..................... 28

4 Numerical methods: higher order 35
4.1 Leapfrog algorithm ............................................. 35
4.2 Analysis of the Leapfrog algorithm ........................ 37
4.2.1 The case \( \alpha \leq 1 \) ......................................... 40
4.2.2 The case \( \alpha > 1 \) ......................................... 43
4.2.3 The general solution ....................................... 43
4.3 Numerical experiments with the Leapfrog algorithm ... 44
4.3.1 Sine wave .................................................. 44
4.3.2 Triangular pulse .......................................... 50
4.3.3 Square pulse ............................................. 53
4.3.4 Gaussian pulse .......................................... 56
4.4 Trapezoid algorithm ......................................... 59
4.5 Analysis of the Trapezoid algorithm ........................ 65
4.6 Solution of the linear equations ............................ 67
4.7 Numerical experiments with the Trapezoid algorithm .... 69
4.7.1 Triangular pulse .......................................... 69
4.7.2 Square pulse ............................................. 70
4.7.3 Gaussian pulse .......................................... 70

CU Boulder: HPSC Course Notes
4.7.4 Comparing the cost of Leapfrog and Trapezoid algorithms

5 Two-dimensional problems
   5.1 Leapfrog and Trapezoid algorithms in two dimensions

6 Conclusion

References
Advection:
An Introduction *

Lloyd Fosdick
September 5, 1995

1 Introduction

When the first automatic computers were built in the middle of this century, weather forecasting was identified as a major problem to be attacked by these new machines. It remains so today. Meteorology, the study of the atmosphere for the purpose of weather forecasting, now has advanced to the state where reasonably accurate five-day forecasts are possible. Climatology, the study of the earth’s climate, is closely related. It requires predicting effects over relatively large time intervals and these predictions are not nearly as reliable as the five-day weather forecast. Indeed, the results of computations predicting global warming have been under dispute. Thus the need for improving the speed and accuracy of these computations continues.

Computations in climatology and meteorology require models of the atmosphere and those aspects of the earth’s surface — mountains, sea, ice, and so forth — that affect the weather and climate. The mathematical description of these models generally takes the form of a complex system of partial differential equations.

*This work has been supported by the National Science Foundation under an Educational Infrastructure grant, CDA-9017953. It has been produced by the HPSC Group, Department of Computer Science, University of Colorado, Boulder, CO 80309. Please direct comments or queries to Elizabeth Jessup at this address or e-mail Jessup@cs.colorado.edu
Copyright ©1995 by the HPSC Group of the University of Colorado
Among these equations is one relatively simple equation, called the *advection equation*, that describes the transport of a substance in the atmosphere. In meteorology the word *convection* refers to the vertical transport of matter due to rising air, and *advection* refers to the horizontal transport of matter due to wind. Thus the advection equation might describe the transport of a moist air mass, a dust cloud, or chemical substances by wind. Studies of large-scale air pollution, as might occur over southern California or northern Europe, represent an important application of this equation.

The advection equation and its solution is the principal subject of this tutorial. There are two good reasons for including it in a course on high-performance scientific computing. In the first place, advection is a part of important applications of high-performance computing: meteorology, climatology, and air pollution. It would go beyond the level of this tutorial to treat the full system of equations for a climate model or an air pollution model, but we can learn something about the computational issues by studying the advection component of these models. Second, many of the computations done on high-performance computers involve the solution of partial differential equations. The advection equation is a relatively simple partial differential equation, so it provides a natural introduction to a broad class of computations performed on supercomputers.

This tutorial consists of a detailed discussion of the one-dimensional advection equation and its solution by several algorithms. These algorithms are illustrated by a number of examples, and a careful analysis of the algorithms is made in order to explain the results. Two algorithms are the focus of most of the discussion, the Leapfrog algorithm and the Trapezoid algorithm. They have been chosen because they are relatively simple and because they serve to illustrate two fundamentally different kinds of numerical methods. The discussion closes with a short section on the two-dimensional advection equation. Here enough material is given to provide the reader the tools necessary to construct and analyze the two-dimensional forms of the Leapfrog and Trapezoid algorithms.

Advection implies the existence of a fluid flow, wind, that is responsible for the transport. Of course the velocity of this flow is not constant but it may be almost constant over relatively long periods. Most of this tutorial assumes constant wind velocity, mainly to keep the introduction to this subject reasonably easy to follow. In the section on the two-dimensional advection equation, a variable velocity is introduced, illustrating the added
Advection

complexity it causes.

It is assumed that you have had a conventional undergraduate course in numerical analysis. It does not assume a course in partial differential equations, but it does assume that you are familiar with partial derivatives and finite difference approximations of derivatives. The mathematical analysis of the algorithms we discuss requires an elementary knowledge of Fourier series. You may wish to read the Numerical Analysis Review [Fosdick & Jessup 95] to refresh your knowledge. If you are interested in background reading in the areas of meteorology and climatology you might consider the books Meteorology [Miller & Anthes 80], An Introduction to Three-Dimensional Climate Modeling [Washington & Parkinson 86], and Numerical Prediction and Dynamic Meteorology [Haltiner & Williams 80]. The first of these should be easily understood by any undergraduate student, the other two are considerably more advanced, the latter has a good discussion of the advection equation and of numerical methods for solving it. Another good source for numerical methods in this area is Numerical Methods Used in Atmospheric Models[Mesinger & Arakawa 76]. The technical report Running Air Pollution Models on the Connection Machine [Zlatev & Wasniewski 92] has an easily understood discussion of air pollution models and the use of a massively parallel computer to study them.
2 One-dimensional advection equation

Imagine a small particle suspended in a fluid flowing with constant velocity \( u \). After an interval of time \( t' \) the particle is carried downstream a distance \( ut' \). Similarly, a group of particles suspended in the fluid are carried along with the flow, each moving a distance \( ut' \), thus their positions relative to each other do not change. Now imagine a very large number of tiny particles suspended in the fluid and let \( \phi(x, t) \) represent their density (the number of particles per unit interval). Figure 1 illustrates this density at time \( t = 0 \) and at a slightly later time \( t = 0.2 \) when \( u = 1 \); the density distribution is simply translated a distance 0.2 in the direction of the flow.

![Density distribution](image)

Figure 1: Example of a density distribution of particles in a one-dimensional flow at time \( t = 0 \) and at a later time \( t = 0.2 \) when \( u = 1 \).

In this model a group of particles in the fluid is carried along by the flow, maintaining their positions relative to each other, therefore it is a trivial matter to determine where every particle is, or what the density distribution of particles is, at a later time. However, when the velocity of the flow is not constant and when the flow is not one-dimensional, this determination cannot be made so easily. Then we must use mathematical equations to represent the physical model and numerical methods to solve them. Nevertheless it is instructive to start our study of the mathematical issues with the simple one-dimensional, constant velocity model. The equation that describes it is
called the *one-dimensional advection equation*, expressed as follows:

\[
\frac{\partial \phi(x,t)}{\partial t} + u \frac{\partial \phi(x,t)}{\partial x} = 0. \tag{1}
\]

The independent variables \(x\) and \(t\) are position and time, respectively; \(\phi(x,t)\) is the density of particles; and \(u\) is the velocity of the flow, which we assume to be constant and in the positive \(x\) direction. In more realistic, more complex models, the right-hand side of this equation is not zero: it may contain terms that represent the effects of diffusion, sources of substances injected into the flow (for example, automobile exhaust), and reactions between chemicals in the flow.

It is customary to use the notation \(\phi_t\) and \(\phi_x\) to denote partial derivatives with respect to \(t\) and \(x\), then the advection equation has the form

\[
\phi_t + u \phi_x = 0.
\]

We can verify that this equation models a distribution of particles carried along by a flow moving at velocity \(u\). Suppose that the initial density of particles is given by \(f(x)\); i.e., \(\phi(x,0) = f(x)\). As time advances the particles are carried along, downstream, so that all of them have moved a distance \(ut\) by time \(t\), as described already. Therefore

\[
\phi(x,t') = \phi(x - ut',0) = f(x - ut').
\]

Thus, if the solution to the advection equation is

\[
\phi(x,t) = f(x - ut), \tag{2}
\]

we know that it correctly represents the particle flow in our simple model. So let us verify that equation (2) is a solution to equation (1) by substituting it into equation (1). For the first term on the left of equation (1) we have, with \(z = x - ut\),

\[
\frac{\partial \phi(x,t)}{\partial t} = \frac{\partial f(z)}{\partial t} = \frac{df(z)}{dz} \frac{dz}{dt} = \frac{df(z)}{dz}(-u);
\]

and for the second term on the left of equation (1) we have

\[
\frac{\partial \phi(x,t)}{\partial x} = \frac{\partial f(z)}{\partial x} = \frac{df(z)}{dz} \frac{dz}{dx} = \frac{df(z)}{dz}u;
\]

therefore

\[
\frac{\partial \phi(x,t)}{\partial t} + u \frac{\partial \phi(x,t)}{\partial x} = \frac{df(z)}{dz}(-u + u) = 0,
\]

thus equation (1), the advection equation, is satisfied.
2.1 Advection equation is $d\phi/dt = 0$

Here is another view of what the advection equation represents. Imagine that you could shrink to the size of a particle and move along with the flow. The change you would see in $\phi$ with time is given by the total derivative $d\phi/dt$ and from elementary calculus we know that

$$\frac{d\phi}{dt} = \phi_t + \frac{dx}{dt}\phi_x.$$  

Comparing this with the left-hand side of the advection equation and noting that $dx/dt = u$ we see that the advection equation is equivalent to the equation

$$\frac{d\phi}{dt} = 0.$$  

It is important to recognize that this result for $d\phi/dt$ holds in a frame of reference that is moving with the flow. In fluid dynamics this frame of reference is called Lagrangian, while a frame of reference that is fixed is called Eulerian.

2.2 Advection equation as a conservation law

Assume a collection of particles with density $\phi(x,t)$ is flowing with velocity $u(x,t)$. Now consider an infinitesimal interval of length $2dx$, centered at $x = a$. If particles are not spontaneously created or destroyed, then the number of particles entering the interval minus the number of particles leaving it must equal the net change in the number of particles in the interval. The number entering and leaving per unit time is

$$N_{\text{enter}} = u(a - dx,t)\phi(a - dx,t)$$

$$N_{\text{leave}} = u(a + dx,t)\phi(a + dx,t)$$

and the net increase in the number of particles per unit time in the interval is

$$N_{\text{increase}} = \frac{\partial \phi(a,t)}{\partial t} \bigg|_{x=a} \, dx.$$  

Conservation of particles requires

$$N_{\text{enter}} - N_{\text{leave}} = N_{\text{increase}}.$$  

CU Boulder: HPSC Course Notes
Advection

\[ u(a - dx, t)\phi(a - dx, t) - u(a + dx, t)\phi(a + dx, t) = \frac{\partial \phi(a, t)}{\partial t}2dx, \]

or

\[ \frac{u(a - dx, t)\phi(a - dx, t) - u(a + dx, t)\phi(a + dx, t)}{2dx} = \frac{\partial \phi(a, t)}{\partial t}. \]

If we now take the limit \( dx \to 0 \) this becomes

\[ - \frac{\partial u(x, t)\phi(x, t)}{\partial x} \bigg|_{x=a} = \frac{\partial \phi(a, t)}{\partial t}. \]

Now use the fact that \( a \) can be any point on the \( x \)-axis and rearrange terms, then

\[ \frac{\partial \phi(x, t)}{\partial t} + \frac{\partial u(x, t)\phi(x, t)}{\partial x} = 0, \quad (3) \]

which is the advection equation when we assume that the velocity can depend on \( x \) and \( t \). If the velocity is constant then this becomes the advection equation as already given, equation (1). Equation (3) is important to remember because it indicates how formulas for the constant velocity case, appearing later, must be modified in order to account for a variable velocity.

This derivation is also important because it suggests how the advection equation must be modified if particles are being created, as would be necessary in a model that takes into account the continual injection of a polluting substance. A nonzero term on the right of this equation would be required to represent this source of particles.

2.3 Boundary conditions

Usually we are interested in knowing the solution to the advection equation in a finite region of the \( x-t \) plane. Here we assume that this region is the rectangle \( R = [0, X] \times [0, T] \), as illustrated in figure 2.

Restricting our attention to the rectangle \( R \) suggests that we specify the initial condition

\[ \phi(x, 0) = f(x) \]

on the interval \([0, X]\). However, this does not determine the solution everywhere in \( R \). We can see this in the following way. In our earlier discussion
we verified that the solution is given by equation (2). It follows from this equation that $\phi(x, t)$ is constant on the straight line $x = ut + x_0$ and on this line $\phi(x, t) = f(x_0)$, where $x_0$ can be any point in $[0, X]$: this line is called a characteristic. Three characteristics are illustrated in figure 3, corresponding to $x_0 = 0.2, 0.4$, and 0.6. Thus for every point on the $x$-axis where $f(x)$ is defined we have such a characteristic and if $f(x)$ is known on an interval, say $[0, X]$, then the associated characteristics sweep out a region as indicated in figure 4. Thus it is evident that there is a region of $R$, bounded on the left by the $t$-axis and on the right by the characteristic through the origin, in which the solution is completely undetermined.

We could resolve this problem by specifying the value of $\phi(x, 0)$ on a larger interval. Our previous discussion should make it evident that if $\phi(x, 0)$ is specified on the interval $[-uT, X]$ then the solution is determined everywhere in $R$. But this is not very satisfactory because we must use points outside the region $R$.

Here is a better way to deal with the problem. If we specify the value of $\phi(0, t)$ on the $t$-interval $[0, T]$ and on the $x$- interval $[0, X]$ then the solution is determined everywhere in $R$. To illustrate, suppose that we know $\phi(0, t')$
Figure 3: Three characteristics: (a) $\phi(x,t) = f(0.2)$, (b) $\phi(x,t) = f(0.4)$, (c) $\phi(x,t) = f(0.6)$.

Figure 4: The shaded region is swept out by the characteristics and in this region the solution is determined. In the unshaded region, bounded on the left by the $t$-axis and on the right by the characteristic through the origin, the solution is completely undetermined.
where \( t' \in [0, T] \), then \( \phi(x, t) \) is determined along the line

\[
t = \frac{1}{u} x + t'.
\]

This follows from the fact that the solution to the advection equation is constant along the characteristics, straight lines with slope \( 1/u \); therefore if we specify the solution at any point on one of these lines it is known everywhere else on that line. Thus if we specify the value of \( \phi(0, t) \) on \( [0, T] \) and \( \phi(x, 0) \) on \( [0, X] \) then the solution is determined at every point in \( R \).

How should we choose \( \phi(0, t) \)? A common choice, and the one we adopt here, is periodic boundary conditions; that is, we require

\[
\phi(0, t) = \phi(X, t), \quad t \in [0, T].
\]

We can see that periodic boundary conditions determine \( \phi(0, t) \) for \( t \in [0, T] \) in the following way. From the characteristics it is seen that the value of \( \phi(X, t) \) for \( t \in [0, X/u] \) is completely determined by \( \phi(x, 0) \); similarly, the value of \( \phi(X, t) \) for \( t \in [X/u, 2X/u] \) is completely determined by \( \phi(x, X/u) \), etc. The use of periodic boundary conditions is equivalent to assuming that the region \( R \) is wrapped on the surface of a cylinder, with the \( t \)-axis parallel to the axis of the cylinder; or, equivalently, to extending the region \( R \) periodically along the \( x \)-axis from \(-\infty \) to \( +\infty \).

Before closing this section we must say some additional words about characteristics. In general they are not straight lines as they are here, nor is the solution constant along them. They do have the following important property: if initial data are provided along a curve \( C \) in the problem space, and this curve is not a characteristic, then that data determines the solution along any characteristic intersecting \( C \); on the other hand, if \( C \) is a characteristic then the initial data is not sufficient to determine the solution elsewhere in the problem space. Note, for example, that if we specified the value of \( \phi \) along a characteristic, its value elsewhere in \( R \) is completely undetermined.

Characteristics are important not only to the theory of partial differential equations, they also provide a means for solving partial differential equations — the solution along a characteristic can be determined by integrating an ordinary differential equation (in our case it is \( d\phi/dt = 0 \)). Not all partial differential equations have characteristics; there are three broad classes of partial differential equations — hyperbolic, parabolic, and elliptic — only the first
two have characteristics. In fact the notion of characteristics is intimately
connected with this classification. For more on this subject see the books Nu-
merical Solution of Partial Differential Equations: Finite Difference Methods
[Smith 78] and Partial Differential Equations [Carrier & Pearson 76].
3 Numerical methods: first order

Here we turn our attention to some simple numerical methods for solving the advection equation. We focus on finite difference algorithms, so named because they are based on approximating the partial derivatives in the advection equation by finite differences. Here we consider algorithms based on first-order approximations of the partial derivatives. While these algorithms are not accurate enough for serious computational work, they provide a convenient tool for introducing ideas we use later in discussing more accurate finite difference methods.

3.1 Discretization of $R$

Finite-difference methods generate the solution on the points of a regularly spaced grid denoted by $R_g$, figure 5. The values of $x$ and $t$ at the grid point $(j,n)$ are $x_j$ and $t_n$:

\[ x_j = j\Delta x, \quad \text{and} \quad t_n = n\Delta t; \]

in particular,

\[ x_0 = 0, \quad x_J = X, \quad t_0 = 0, \quad \text{and} \quad t_N = T. \]

Thus the continuous region $R$ is discretized and replaced by the region $R_g$, also referred to as “the grid”.

The solution $\phi$ at $(j,n)$ is denoted by $\phi(x_j,t_n)$ or equivalently by $\phi_{j,n}$. For brevity, we often use $\phi_{j,n}$. The notation $\phi_{0,:}$ denotes the set of values $\phi_{0,n}, \phi_{1,n}, \ldots, \phi_{J,n}$, the solution at all values of $x$ at time $t_n$; similarly, $\phi_{0,:0:N}$ denotes the solution on the entire region $R_g$.

In $R_g$ periodic boundary conditions are expressed with

\[ \phi_{0,0:N} = \phi_{J,0:N}, \]

and the initial condition is expressed with

\[ \phi_{0,J-1,0} = f(x_{0,J-1}). \]

Periodic boundary conditions make it unnecessary to specify $\phi_{J,0}$. 

CUBoulder: HPSC Course Notes
Advection

3.2 First-order difference methods

Recall from Taylor’s series that
\[
\phi(x + \Delta x, t) = \phi(x, t) + \phi_x(x, t) \Delta x + O((\Delta x)^2),
\]
\[
\phi(x, t + \Delta t) = \phi(x, t) + \phi_t(x, t) \Delta t + O((\Delta t)^2).
\]

Therefore,
\[
\phi_x(x, t) = \frac{\Delta x}{\Delta x} \frac{\phi(x + \Delta x, t) - \phi(x, t)}{\Delta x} + O(\Delta x),
\]
\[
\phi_t(x, t) = \frac{\Delta t}{\Delta t} \frac{\phi(x, t + \Delta t) - \phi(x, t)}{\Delta t} + O(\Delta t). \tag{4}
\]

The expressions \(\phi(x + \Delta x, t) - \phi(x, t)\) and \(\phi(x, t + \Delta t) - \phi(x, t)\) appearing here are called “forward differences”.

3.2.1 A forward difference algorithm, Algorithm 1

If we use equations (4, 4) to replace the partial derivatives in the advection equation we obtain
\[
\frac{\phi_{j,n+1} - \phi_{j,n}}{\Delta t} + u \frac{\phi_{j+1,n} - \phi_{j,n}}{\Delta x} + O(\Delta x) + O(\Delta t) = 0.
\]
Advection

Then if we drop the terms $O(\Delta x)$ and $O(\Delta t)$ we obtain a finite difference approximation of the advection equation:

$$\frac{\phi_{j,n+1} - \phi_{j,n}}{\Delta t} + u \frac{\phi_{j+1,n} - \phi_{j,n}}{\Delta x} = 0. \quad (5)$$

The numerical method we discuss now computes a solution to this equation. Rearrange the terms in equation (5) to obtain

$$\phi_{j,n+1} = \phi_{j,n} - \alpha (\phi_{j+1,n} - \phi_{j,n}), \quad (6)$$

where

$$\alpha = u \frac{\Delta t}{\Delta x}.$$ 

Here you can see that we have the basis of an algorithm for computing the solution at time $t_{n+1}$ in terms of the solution at time $t_n$. Since the initial condition provides the value of $\phi_{0,J-1,0}$ we can compute the value of $\phi_{0,J-1,1}$ with equation (6), then the value of $\phi_{0,J-1,2}$, and so on. Thus we have Algorithm 1.

[Algorithm 1]

\begin{verbatim}
[Algorithm 1]
\phi_{0,J-1,0} = f(x_{0,J-1})
\text{for } n = 0 : N - 1
  \phi_{J,n} = \phi_{0,n}
  \phi_{0,J-1,n+1} = \phi_{0,J-1,n} - \alpha (\phi_{1,J,n} - \phi_{0,J-1,n})
\end{verbatim}

One detail of this algorithm deserves special comment. The update of the value of $\phi$ at the right boundary, $j = J - 1$, is special because of the periodic boundary conditions; for example,

$$\phi_{J-1,n+1} = \phi_{J-1,n} - \alpha (\phi_{0,n} - \phi_{J-1,n}).$$

Notice that the value of $\phi_{0,n}$ is used on the right side of this expression, so this update does not have the same pattern as the update at the other points, $j = 0, 1, \ldots, J - 2$. However, by copying the value $\phi_{0,n}$ to the point $(J, n)$ (see the first statement in the loop) the update of the right boundary
Advection

Figure 6: The triangular pulse used to specify the initial condition for Algorithm 1.

Point is done within the single vector statement of the time-stepping loop. Not only does this make the code simpler in appearance, it has the potential for producing a faster computation on a machine with vector arithmetic.

3.2.2 Test of the forward-difference algorithm, Algorithm 1

Unfortunately Algorithm 1 is far from satisfactory as we now see with an example. We use a triangular pulse for the initial condition:

\[
f(x) = \begin{cases} 
0 & \text{if } 0 \leq x \leq 0.25; \\
4x - 1 & \text{if } 0.25 < x \leq 0.5; \\
3 - 4x & \text{if } 0.5 < x \leq 0.75; \\
0 & \text{if } 0.75 < x \leq 1.
\end{cases}
\]  

This function is illustrated in figure 6.

The computed and exact solutions at \(t = 0.1\) for \(u = 0.4\) and \(\Delta t = \Delta x = 0.01\) are shown in figure 7. Since \(\Delta t = 0.01\) this is the result after ten iterations, \(N = 10\) in Algorithm 1. The computed solution at \(t = 0.2\) is shown in figure 8. In this second picture the computed solution has grown by two orders of magnitude, so large that the exact solution appears to be on the x-axis in the scale of this figure. Clearly this algorithm is producing
Figure 7: The solution computed by Algorithm 1, marked by “o”, and the exact solution, marked by “+”, at $t = 0.1$.

completely unsatisfactory results after just twenty time steps. The reason for this behavior is explained shortly, but first we look at a slightly different algorithm that gives much better results.

### 3.2.3 A backward difference algorithm, Algorithm 2

Again from Taylor’s series we have

$$\phi(x - \Delta x, t) = \phi(x, t) - \phi_x(x, t)\Delta x + \mathcal{O}((\Delta x)^2),$$

from which we obtain

$$\phi_x(x, t) = \frac{\phi(x, t) - \phi(x - \Delta x, t)}{\Delta x} + \mathcal{O}(\Delta x). \quad (8)$$

The expression $\phi(x, t) - \phi(x - \Delta x, t)$ appearing here is called a “backward difference”.

If we use this new formula, equation (8), for $\phi_x(x, t)$ and the old formula, equation (4), for $\phi_t(x, t)$ in the advection equation we obtain

$$\frac{\phi_{j,n+1} - \phi_{j,n}}{\Delta t} + u \frac{\phi_{j,n} - \phi_{j-1,n}}{\Delta x} + \mathcal{O}(\Delta x) + \mathcal{O}(\Delta t) = 0.$$
As before, we drop the terms $\mathcal{O}(\Delta x)$ and $\mathcal{O}(\Delta t)$ and arrive at a new formula for computing $\phi_{j,n+1}$:

$$\phi_{j,n+1} = \phi_{j,n} - \alpha(\phi_{j,n} - \phi_{j-1,n}) .$$

Thus we have:

---

**Algorithm 2**

$$\begin{align*}
\phi_{0,J-1,0} &= f(x_{0,J-1}) \\
\text{for } n &= 0 : N - 1 \\
\phi_{-1,n} &= \phi_{J-1,n} \\
\phi_{0:J-1,n+1} &= \phi_{0:J-1,n} - \alpha(\phi_{0:J-1,n} - \phi_{-1:J-2,n})
\end{align*}$$

Notice that we have again used the bordering idea: in this case for the update of $\phi_{0,n}$.
3.2.4 Test of the backward difference algorithm, Algorithm 2

To illustrate the performance of this algorithm we again choose the triangular pulse, figure 6, for the initial condition. The computed solution at $t = 0.1$ and $t = 0.2$ for $u = 0.4$ and $\Delta t = \Delta x = 0.01$ is shown in figures 9 and 10. It is evident that this algorithm produces much better results. Figure 11 shows the computed solution after a much longer time, $t = 10$. Notice that the shape of the pulse is damped and spread out quite a bit but the peak is in about the right position: the peak of the exact solution is at $x = 0.5$ because the velocity of the pulse is 0.4 so it will have moved a distance of 4 at $t = 10$, thus returning to its initial position because of the periodic boundary conditions.

3.2.5 Analysis of Algorithms 1 and 2

We use a well-known technique due to von Neumann for analyzing the two methods just described. It assumes that the solution is represented as a Fourier series

$$
\phi(x, t) = c_0 + \sum_{\nu=1}^{\infty} (c_{\nu} \cos(2\pi \nu (x - ut)) + s_{\nu} \sin(2\pi \nu (x - ut))).
$$

(10)
Figure 10: The solution computed by Algorithm 2, marked by “o”, and the exact solution, marked by “+”, at $t = 0.2$.

Figure 11: The solution computed by Algorithm 2, marked by “o”, and the exact solution, marked by “+”, at $t = 10.0$. 
This analysis shows how the algorithms act on individual terms in the Fourier series and from this we are able to understand their effect on the triangular pulse and other functions. We should remind you that in this context \( u \) is referred to as the \textit{phase velocity} of the wave represented by the sine or cosine term in which it appears.

The triangular pulse we have been using is represented by a Fourier series with coefficients:

\[
c_{\nu} = \begin{cases} 
1/4 & \text{if } \nu = 0; \\
-4/(\nu \pi)^2 & \text{if } \nu \text{ is odd;} \\
0 & \text{if } \nu \text{ is divisible by 4;} \\
8/(\nu \pi)^2 & \text{otherwise}
\end{cases}; \hspace{1cm} s_{\nu} = 0. \tag{11}
\]

For example, taking terms up to \( \nu = 7 \) we have the approximation

\[
\phi(x, t) \approx \frac{1}{4} + \frac{1}{\pi^2} (-4 \cos(2\pi(x - ut)) + \frac{8}{4} \cos(4\pi(x - ut)) \\
- \frac{4}{9} \cos(6\pi(x - ut)) - \frac{4}{25} \cos(10\pi(x - ut)) \\
+ \frac{8}{36} \cos(12\pi(x - ut)) - \frac{4}{49} \cos(14\pi(x - ut)).
\]

Figure 12 shows a plot of this Fourier series approximation to the triangular pulse using terms up to \( \nu = 15 \), when \( t = 0 \).

For the analysis that follows it is more convenient to put the Fourier series into exponential form:

\[
\phi(x, t) = \sum_{\nu=-\infty}^{+\infty} a_{\nu} e^{i\omega(x - ut)}.
\]

For brevity we use \( \omega = 2\pi \nu \).

The analysis proceeds by examining the effect of the two algorithms on a single Fourier component, \( e^{i\omega(x - ut)} \). The real part of this component represents a cosine wave with frequency \( \nu \) and phase velocity \( u \); and the imaginary part a sine wave with frequency \( \nu \) and phase velocity \( u \). By determining the effect of the algorithm on a single Fourier component we are able to infer its effect on the entire Fourier series, and thus its effect on the solution to the advection equation. The following analysis could deal explicitly with the cosine or sine waves but the algebra is simpler if we use the exponential \( e^{i\omega(x - ut)} \).
We choose $X = 1$ so that the region $R$ is the rectangle $[0, 1] \times [0, T]$; thus in $R_g$ we have $J \Delta x = 1$ and $N \Delta t = T$. It should be evident from the earlier discussion that if

$$\phi(x, 0) = e^{i \omega x},$$

then the solution to the advection equation is

$$\phi(x, t) = e^{i \omega (x - ut)},$$

the single Fourier component we will consider. Notice that it does satisfy the initial condition equation (12) and the periodic boundary conditions.

Now compare the solution generated by Algorithm 1 with this exact solution. For the initial condition we have

$$\phi_{0, J-1, 0} = e^{i \omega x_{0; J-1}},$$

and after one time step

$$\phi_{0, J-1, 1} = \phi_{0, J-1, 0} - \alpha (e^{i \omega \Delta x} \phi_{0, J-1, 0} - \phi_{0, J-1, 0}),$$

which can be written

$$\phi_{0, J-1, 1} = A \phi_{0, J-1, 0},$$
where
\[ A = 1 - \alpha(e^{i\omega\Delta x} - 1). \] (14)

Thus the effect of a time step, one iteration of Algorithm 1, is to multiply the solution at the present time step by \( A \). Therefore, after \( n \) time steps the computed solution is
\[ \phi_{0,J-1,n} = A^n\phi_{0,J-1,0}. \] (15)

Notice that the exact solution can be expressed in exactly the same form:
\[ \phi_{0,J-1,n}^{(\text{exact})} = B^n\phi_{0,J-1,0}, \]
where
\[ B = e^{-i\omega u\Delta t}. \]

Thus the difference between the computed and exact solutions is contained in the difference between \( A \) and \( B \).

In order to see this difference clearly we put the complex number \( A \) into polar form, that is
\[ A = \rho e^{i\theta}. \] (16)

If the computed solution were equal to the exact solution then \( \rho = 1 \) and \( \theta = -\omega u\Delta t \). Notice that \( \rho \) determines whether the amplitude of the computed solution increases with time or decreases with time: \( \rho > 1 \) implies increasing amplitude, \( \rho < 1 \) implies decreasing amplitude, and \( \rho = 1 \) implies that the amplitude does not change.

Substitution of \( A \), given by equation (16), and \( \phi_{0,J-1,0} \), given by equation (13), into equation (15) gives
\[ \phi_{0,J-1,n} = \rho^n e^{i\omega(x_{J-1} + \theta - \omega u\Delta t)}. \]

From this formula it is evident that the phase velocity of a wave in the computed solution is
\[ u^{(\text{computed})} = -\frac{\theta}{\omega\Delta t}. \] (17)

Considering the real and imaginary parts of \( A \) in equation (14) and some simple algebra we can find \( \rho \) and \( \theta \) of \( A \) in equation (16):
\[ \rho^2 = 1 + 2\alpha(1 + \alpha)(1 - \cos(\omega\Delta x)), \] (18)
Figure 13: The amplification factor $\rho$ of Fourier components for a solution computed by Algorithm 1 shown as a function of frequency, $\nu$, when $u = 0.4$ and $\Delta t = \Delta x = 0.01$.

and

$$\theta = \arctan \left( -\frac{\alpha \sin(\omega \Delta x)}{1 - \alpha (\cos(\omega \Delta x) - 1)} \right).$$

Therefore

$$\rho \geq 1$$

because $\alpha > 0$ and $(1 - \cos(\omega \Delta x)) \geq 0$; and

$$u^{(\text{computed})} = \arctan \left( \frac{\alpha \sin(\omega \Delta x)}{1 - \alpha (\cos(\omega \Delta x) - 1)} \right) \frac{1}{\omega \Delta t}. \quad (19)$$

We see here that the amplitude of every Fourier component increases with time (excepting the special case where $1 - \cos(\omega \Delta x) = 0$). This explains the very large values of the computed solution after just 20 time steps shown in figure 8. Figure 13 shows the amplification factor $\rho$ as a function of frequency when $u = 0.4$ and $\Delta t = \Delta x = 0.01$. Notice that the higher frequency components are amplified the most; since these are least important so far as the general shape of the exact solution is concerned, their amplification causes a serious distortion of the computed solution.

The dependence of the phase velocity on frequency, equation (19) is made evident by a graph, shown in figure 14. We see here that the phase velocity
Figure 14: The phase velocity for the solution computed by Algorithm 1 shown as a function of frequency, $\nu$, when $u = 0.4$ and $\Delta t = \Delta x = 0.01$.

is less than the correct value (0.4) at all frequencies, with the low frequency waves having velocities closest to the correct value. The word “dispersion” is used to describe the fact that waves of different frequency travel at different speeds — the word derives from the fact that a travelling wave form such as the triangular pulse spreads out (disperses) when its Fourier components are travelling at different speeds.

The important conclusion we draw from this analysis is that the error in the computed solution is influenced by two factors — growth in amplitude of the Fourier components and dispersion. This error should decrease as $\Delta x \to 0$ and $\Delta t \to 0$. To confirm this expectation we look at the values of $\rho$ and $\theta$ in this limit. In going to the limit it is useful to hold $\alpha$ constant, which means that we must keep the ratio $\Delta t/\Delta x$ constant. From equation (18) and a Taylor series expansion of $\cos(\omega \Delta x)$ we have

$$\rho^2 = 1 + 2\alpha(1 + \alpha)(\omega \Delta x)^2/2 + \mathcal{O}((\Delta x)^4);$$

and from equation (19) we have

$$u^{(\text{computed})} = \alpha \omega \Delta x \frac{1}{\omega \Delta t} + \mathcal{O}((\Delta x)^2),$$

$$= u + \mathcal{O}((\Delta x)^2).$$
Advection

Thus we can conclude that in the limit

\[ \Delta x \to 0, \; \Delta t \to 0, \; \text{with} \; \frac{\Delta t}{\Delta x} \text{ constant,} \]

we have

\[ \rho \to 1, \; u^{(\text{computed})} \to u. \]

Let us now turn our attention to Algorithm 2. Having introduced the basic ideas of the analysis already we go immediately to the results. In this case we find

\[ \phi_{0,J-1,n} = A^n \phi_{0,J-1,0}, \] where now \( A = 1 - \alpha(1 - e^{-i\omega \Delta x}) \).

Hence

\[ \rho^2 = 1 - 2\alpha(1 - \alpha)(1 - \cos(\omega \Delta x)). \] (20)

We can distinguish three cases:

\[ \rho^2 \begin{cases} < 1 & \text{if } \alpha < 1, \; \omega \Delta x \neq 0; \\ = 1 & \text{if } \alpha = 1; \\ > 1 & \text{if } \alpha > 1. \end{cases} \]

Therefore the amplitude of the Fourier components (excepting the zero frequency component) decreases with time if \( \alpha < 1 \); remains constant if \( \alpha = 1 \); and increases with time if \( \alpha > 1 \). In fact when \( \alpha = 1 \) the computed solution is the exact solution, as can be seen by replacing \( \alpha \) by 1 in equation (9).

The phase velocity in the computed solution is given by equation (17) where now

\[ \theta = \arctan \left( \frac{-\alpha \sin \omega \Delta x}{1 - \alpha(1 - \cos \omega \Delta x)} \right). \] (21)

To verify your understanding of this show that the computed velocity is equal to the exact velocity when \( \alpha = 1 \) using equation (17) and equation (21).

We again illustrate the dependence of \( \rho \) and the phase velocity on frequency: \( \rho \) is shown as a function of frequency in figure 15; the phase velocity is shown as a function of frequency in figure 16. In both of these figures \( u = 0.4 \) and \( \Delta t = \Delta x = 0.01. \)
Figure 15: The amplification factor $\rho$ of Fourier components for the solution computed by Algorithm 2 shown as a function of frequency, $\nu$, when $u = 0.4$ and $\Delta t = \Delta x = 0.01$.

Figure 16: The phase velocity for the solution computed by Algorithm 2 shown as a function of frequency, $\nu$, when $u = 0.4$ and $\Delta t = \Delta x = 0.01$. 

CUBoulder : HPSC Course Notes
Advection

Notice that the decrease in amplitude is most significant for the higher frequencies which are least important for the general shape of the solution. Figure 12 shows, for example, that the shape of the triangle pulse is well represented by the first fifteen frequencies. Also, the effect of dispersion is reduced because it is the high frequency components, which are damped, that are travelling significantly more slowly. Thus the general shape of the solution is better preserved in Algorithm 2 than in Algorithm 1.

An analysis like that done for Algorithm 1 shows that Algorithm 2 has the same limiting behavior; that is, \( \rho \to 1 \) and \( u^{(\text{computed})} \to u \) when \( \Delta x \to 0 \) and \( \Delta t \to 0 \) with \( \Delta t/\Delta x \) constant. Again, you might confirm your understanding of this analysis by confirming this claim.

We might have expected Algorithm 2 would give more accurate results than Algorithm 1 by a loose qualitative argument based on characteristics. Figure 17 shows the grid points used to compute a value of the solution at the next time level. The computation in Algorithm 1 uses a weighted average of \( \phi \) at points \( b \) and \( c \) to determine the value of \( \phi \) at point \( d \), and Algorithm 2 uses a weighted average of the values of \( \phi \) at points \( a \) and \( b \) to determine the value of \( \phi \) at point \( d \). The line passing through point \( d \) in the figure represents a characteristic, therefore we know that the value of \( \phi \) all along this line is the value of \( \phi \) at point \( d \). In particular, at the point where the characteristic passes between \( a \) and \( b \), \( \phi \) has the same value that it has at \( d \). Therefore, since \( a \) and \( b \) lie closer to this point than the pair \( b \) and \( c \) it is reasonable to expect that a weighted average of values of \( \phi \) at \( a \) and \( b \) (Algorithm 2) would give the more accurate estimate of \( \phi \) at \( d \).

The technique employed in Algorithm 2, wherein a backward difference is used so that the grid point at \((i-1,n)\) is used in approximating the partial derivative with respect to \( x \) at the grid point \((i,n)\), is called upwinding. This name derives from the fact that the point \((i-1,n)\) is upwind, with respect to the flow velocity, from the point \((i,n)\); note that \( u > 0 \) implies that the fluid is flowing from \((i-1,n)\) to \((i,n)\).

An algorithm is called unstable when it causes growth in the amplitude of the Fourier components of a solution. Thus we say that Algorithm 1 is unstable, and Algorithm 2 is unstable if \( \alpha > 1 \), otherwise it is stable. An algorithm is said to be dissipative if it causes a decrease in the amplitude of the Fourier coefficients. Thus we say that Algorithm 2 is dissipative if \( \alpha < 1 \).

It is quite evident that the dimensionless parameter \( \alpha \) plays an important
role in determining the characteristics of a solution. It is sometimes called the Courant number in recognition of the work by Courant, Friedrichs, and Lewy [Courant et al 67] in which the concept of stability was introduced. The condition $\alpha \leq 1$ for stability is called the "von Neumann condition".

### 3.3 Confirmation of analysis of Algorithm 2

Having analyzed in detail the effect of Algorithm 2 on a Fourier component we now confirm this analysis with some numerical experiments. Because of periodic boundary conditions for $x \in [0, 1]$, our given domain, we know that the exact solution of the advection equation satisfies the equation

$$
\phi(x, n/u) = \phi(x, 0), \quad n = 0, 1, 2, \ldots;
$$

that is, $\phi$ returns to its initial value after a time interval of $1/u$. However we know from the analysis above that the amplitude of a Fourier component changes by a factor of $\rho$ at each time step and the velocity of a Fourier component is less than $u$. Therefore, the numerical solution will not satisfy equation (22) and our analysis allows us to predict the difference exactly. It is
Table 1: Values of $\rho$ and phase velocity at two different frequencies, $\nu$, as predicted by analysis of Algorithm 2. Parameter values are $u = 0.4$, $\Delta t = \Delta x = 0.01$.

<table>
<thead>
<tr>
<th>$\nu$</th>
<th>$\rho$</th>
<th>phase velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.999526302628</td>
<td>0.399968405541</td>
</tr>
<tr>
<td>5</td>
<td>0.988183752053</td>
<td>0.399203038709</td>
</tr>
</tbody>
</table>

Table 2: Predicted amplitude of solution (a sine wave) at times $t = 1/u$, $t = 2/u$, and $t = 3/u$ and frequencies $\nu = 1$ and $\nu = 5$ for Algorithm 2. Parameter values are $u = 0.4$, $\Delta t = \Delta x = 0.01$.

<table>
<thead>
<tr>
<th>$\nu$</th>
<th>$t = 2.5$</th>
<th>$t = 5.0$</th>
<th>$t = 7.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.88829497037e+00</td>
<td>0.789066102830e+00</td>
<td>0.700923027804e+00</td>
</tr>
<tr>
<td>5</td>
<td>0.512185418526e-01</td>
<td>0.26233902951e-02</td>
<td>0.13436359877e-03</td>
</tr>
</tbody>
</table>

Instructive to run a few tests in which we compare the results of a numerical solution with the predicted results at times $1/u$, $2/u$, and $3/u$.

In these tests we use a single Fourier component and two different frequencies for the initial condition; namely,

$$\phi_{0,J-1,0} = \sin(2\pi \nu(0 : J - 1) \Delta x), \quad \nu = 1, 5.$$  \hspace{1cm} (23)

We also use $u = 0.4$, and $\Delta x = \Delta t = 0.01$, hence $\alpha = 0.4$. Therefore the times of interest are 2.5, 5.0, and 7.5.

To make our prediction we must first determine $\rho$ and the wave velocity from equations (17, 20, 21). A little arithmetic produces the results shown in table 1.

Now it is easy to predict the amplitude of the solution at each of the three times. For example, since $\Delta t = 0.01$ it takes 250 steps to reach $t = 2.5$; hence the amplitude at this time is $\rho^{250}$. Thus for $\nu = 1$ the amplitude is 0.8882941 at $t = 2.5$. A summary of the predicted amplitudes is given in table 2.

We can also predict the distance the wave will have traveled at $t = 2.5$. For example, if $\nu = 1$ then its velocity is 0.3999684 and so it will have traveled...
Distance

<table>
<thead>
<tr>
<th>$\nu$</th>
<th>$t = 2.5$</th>
<th>$t = 5.0$</th>
<th>$t = 7.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.999921013853</td>
<td>1.999842027706</td>
<td>2.999763041560</td>
</tr>
<tr>
<td>5</td>
<td>0.99807596772</td>
<td>1.996015193545</td>
<td>2.994022790318</td>
</tr>
</tbody>
</table>

Table 3: Predicted distance traveled by the solution at times $t = 1/u$, $t = 2/u$, and $t = 3/u$ and frequencies $\nu = 1$ and $\nu = 5$ for Algorithm 2. Parameter values are $u = 0.4$, $\Delta t = \Delta x = 0.01$.

a distance 0.9999210; or, to put it another way, it will lag behind its correct position by 0.0000790. A summary of the predicted distances travelled is shown in table 3.

Next we compare these predictions with the numerical results. Values of the solution obtained from the execution of Algorithm 2 using the initial condition given above in equation (23), with $\nu = 1$, are shown in table 4. The values are shown in the neighborhood of the peak and the zero in the solution: note that the exact solution would have a peak at $x = 0.25$ and would be zero at $x = 0.5$ when $t = 2.5$, $t = 5.0$, and $t = 7.5$. It is apparent from simply scanning this table that the peak and zero are approximately where they should be.

To confirm the agreement between the results in table 4 with our prediction we can compare the peak value of the solution with the predicted amplitude. This comparison is complicated somewhat because the peak value does not occur exactly at $x = 0.5$ but at a point slightly less than this — our prediction in table 3 would put the peak at 0.499921. Since the slope of the solution is zero at this peak the difference in its value at 0.5 from its value at 0.499921 is very small. Therefore we will ignore this difference, and indeed the computed solution (0.88829398764) at $t = 2.5$ and $x = 0.25$ agrees with the predicted amplitude (table 2) to six significant figures. The same close agreement occurs at $t = 5.0$; the agreement is to five significant figures at $t = 7.5$, the effect of the lag in position becoming slightly more noticeable at this later time.

We can locate the zero by interpolation. For example, using linear interpolation on the values at $x = 0.49$ and $x = 0.50$, $t = 2.5$, we estimate that the zero is at 0.4999210, indicating a lag in position of 0.0000790 as

*Advection*

CUBoulder: HPSC Course Notes
Table 4: Results obtained from the execution of Algorithm 2 with initial condition $\phi(x,0) = \sin(2\pi x)$: note $\nu = 1$. Column 1 is time ($t$), column 2 is position ($x$), column 3 is computed solution ($\phi(x,t)$). Parameter values are $u = 0.4$, $\Delta t = \Delta x = 0.01$. 

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x$</th>
<th>$\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.50E+00</td>
<td>2.40E-01</td>
<td>8.8656882336E-01</td>
</tr>
<tr>
<td>2.50E+00</td>
<td>2.50E-01</td>
<td>8.8829398764E-01</td>
</tr>
<tr>
<td>2.50E+00</td>
<td>2.60E-01</td>
<td>8.8651346138E-01</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>2.50E+00</td>
<td>4.90E-01</td>
<td>5.5336464230E-02</td>
</tr>
<tr>
<td>2.50E+00</td>
<td>5.00E-01</td>
<td>-4.4084665887E-04</td>
</tr>
<tr>
<td>2.50E+00</td>
<td>5.10E-01</td>
<td>-5.6216417728E-02</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>5.00E+00</td>
<td>2.40E-01</td>
<td>7.8755815032E-01</td>
</tr>
<tr>
<td>5.00E+00</td>
<td>2.50E-01</td>
<td>7.8906601414E-01</td>
</tr>
<tr>
<td>5.00E+00</td>
<td>2.60E-01</td>
<td>7.874979489E-01</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>5.00E+00</td>
<td>4.90E-01</td>
<td>4.8764207570E-02</td>
</tr>
<tr>
<td>5.00E+00</td>
<td>5.00E-01</td>
<td>-7.8320287310E-04</td>
</tr>
<tr>
<td>5.00E+00</td>
<td>5.10E-01</td>
<td>-5.0327522372E-02</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>7.50E+00</td>
<td>2.40E-01</td>
<td>6.9960466739E-01</td>
</tr>
<tr>
<td>7.50E+00</td>
<td>2.50E-01</td>
<td>7.0092225094E-01</td>
</tr>
<tr>
<td>7.50E+00</td>
<td>2.60E-01</td>
<td>6.9947361459E-01</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>7.50E+00</td>
<td>4.90E-01</td>
<td>4.2969760017E-02</td>
</tr>
<tr>
<td>7.50E+00</td>
<td>5.00E-01</td>
<td>-1.0435715192E-03</td>
</tr>
<tr>
<td>7.50E+00</td>
<td>5.10E-01</td>
<td>-4.5052784556E-02</td>
</tr>
</tbody>
</table>
Table 5: Predicted value of the solution at times $t = 2.5$, $t = 5.0$, and $t = 7.5$ using predicted amplitudes $A'$ from table 2 and predicted velocity $u'$ from table 1.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$t = 2.5$</th>
<th>$t = 5.0$</th>
<th>$t = 7.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>8.8829398764E-01</td>
<td>7.8906601414E-01</td>
<td>7.0092225094E-01</td>
</tr>
<tr>
<td>0.50</td>
<td>-4.4084665887E-04</td>
<td>-7.8320287309E-04</td>
<td>-1.0435715192E-03</td>
</tr>
</tbody>
</table>

predicted. We can compare the computed and analytical position of the zero for the other times similarly, however there is a more precise way to make all of these comparisons.

We can simply compute the value of $A \sin(2\pi \nu(x - u't))$ at $x = 0.25$, 0.5 and $t = 2.5$, 5.0, 7.5 using the predicted amplitude for $A$ and the predicted phase velocity for $u'$. The results of this computation are in table 5. Comparison of these results with those shown in table 4 show complete agreement to 11 significant figures excepting one case where the agreement is to 10 significant figures a difference that can be attributed to roundoff error.

Results obtained for Algorithm 2 when $\nu = 5$ are shown in table 6. Here the results are tabulated only in the neighborhood of the first peak ($x = 0.05$) and the first zero ($x = 0.10$). The predicted values of the solution are shown at the points of interest in table 7. Again we see that the the results agree to eleven significant figures.
Table 6: Results obtained from the execution of Algorithm 2 with initial condition 
\( \phi(x,0) = \sin(10\pi x) \): note \( \nu = 5 \). Column 1 is time \( t \), column 2 is position \( x \), column 3 is computed solution \( \phi(x,t) \). Parameter values are \( u = 0.4, \Delta t = \Delta x = 0.01 \).
Table 7: Predicted value of the solution near the first peak and near the first zero at times $t = 2.5$, $t = 5.0$, and $t = 7.5$, using predicted amplitudes $A'$ from table 2 and predicted velocity $u'$ from table 1.
4 Numerical methods: higher order

Here we consider more accurate finite difference algorithms for solving the advection equation. The first of these is a popular scheme known as the Leapfrog algorithm, the second is sometimes called the Trapezoid algorithm. An important difference between these two algorithms is that Leapfrog is an explicit method, meaning the solution at a new time point is obtained by simply evaluating an expression involving values of the solution at an earlier time, or times, just as in the first-order algorithms introduced in the last section. On the other hand the Trapezoid algorithm is an implicit method, meaning that it is necessary to solve a system of equations in order to determine the solution at a new time point. Thus an implicit method requires more work at each time step. However, as we see later, an implicit method may be competitive with an explicit method.

4.1 Leapfrog algorithm

This algorithm uses second-order approximations to the derivatives rather than the first-order approximations used in Algorithm 1 and Algorithm 2 of the last section. Proceeding as before from Taylor’s series we have

\[
\phi(x + \Delta x, t) = \phi(x, t) + \phi_x(x, t)\Delta x + \frac{1}{2}\phi_{xx}(x, t)(\Delta x)^2 + \mathcal{O}((\Delta x)^3),
\]

\[
\phi(x - \Delta x, t) = \phi(x, t) - \phi_x(x, t)\Delta x + \frac{1}{2}\phi_{xx}(x, t)(\Delta x)^2 - \mathcal{O}((\Delta x)^3).
\]

Subtracting the second equation from the first we have

\[
\phi_x(x, t) = \frac{\phi(x + \Delta x, t) - \phi(x - \Delta x, t)}{2\Delta x} + \mathcal{O}((\Delta x)^2).
\]

If the term \(\mathcal{O}((\Delta x)^2)\) is deleted we have the central difference approximation of \(\phi_x(x, t)\).

In a similar way we can obtain the following expression for the partial derivative with respect to \(t\):

\[
\phi_t(x, t) = \frac{\phi(x, t + \Delta t) - \phi(x, t - \Delta t)}{2\Delta t} + \mathcal{O}((\Delta t)^2),
\]

and if the term \(\mathcal{O}((\Delta t)^2)\) is deleted, we have the central difference approximation of \(\phi_t(x, t)\).
Advection

Substitution of these expressions for the partial derivatives into the advection equation gives
\[
\frac{\phi(x, t + \Delta t) - \phi(x, t - \Delta t)}{2\Delta t} + u^i \frac{\phi(x + \Delta x, t) - \phi(x - \Delta x, t)}{2\Delta x} + \mathcal{O}((\Delta x)^2) + \mathcal{O}((\Delta t)^2) = 0.
\]

Now if we drop the terms \(\mathcal{O}((\Delta x)^2)\) and \(\mathcal{O}((\Delta t)^2)\) and confine \(x\) and \(t\) values to the grid space \(R_g\) we obtain this finite difference approximation to the advection equation:
\[
\frac{\phi_{j, n+1} - \phi_{j, n-1}}{2\Delta t} + u^i \frac{\phi_{j+1, n} - \phi_{j-1, n}}{2\Delta x} = 0. \tag{24}
\]

Rearranging terms in this equation to give
\[
\phi_{j, n+1} = \phi_{j, n-1} + \alpha (\phi_{j+1, n} - \phi_{j-1, n}) , \tag{25}
\]
we have the basic formula for the Leapfrog algorithm. Its name derives from the fact that in computing \(\phi_{j, n+1}\) the algorithm ignores (leaps over) the grid point \((j, n)\), the point at which the partial derivatives are being approximated in equation (24). The template used in the update is shown in figure 18. To verify your understanding of this formula you might try deriving it under the assumption that the velocity is not constant, using equation (3).

Since equation (25) is based on a finite difference approximation to the advection equation having an error that is second order in \(\Delta t\) we say that the accuracy of equation (25), and the Leapfrog algorithm, is second order. See the book by Richtmyer and Morton [Richtmyer & Morton 67, pages 67,68] for a more detailed explanation of "order of accuracy" of finite difference approximations to partial differential equations.

Equation (25) requires values at two time levels, \(n\) and \(n - 1\), in order to compute the solution at the next time level, \(n + 1\). It is apparent that something special is required to start an algorithm based on this equation because the initial condition specifies the value of \(\phi\) at only one time level, \(n = 0\). A simple solution to this problem uses a forward difference approximation for \(\phi_t\) in the first step, keeping the central difference approximation for \(\phi_x\). This leads to the following formula for the first time step:
\[
\phi_{j, 1} = \phi_{j, 0} - \frac{\alpha}{2} (\phi_{j+1, 0} - \phi_{j-1, 0}) . \tag{26}
\]
The penalty for using this formula is a slightly larger error in the first step, and it is reasonable to expect that the overall accuracy of the computation is not significantly affected by this.

Thus we arrive at the Leapfrog algorithm shown in figure 19 that is based on equations (25, 26). Note that the amount of arithmetic required in each time step of this algorithm is the same as for Algorithms 1 and 2.

Figures 20 and 21 show the computed solution and the exact solution at $t = 0.2$ and $t = 10$, using the same triangular pulse, figure 6, for the initial values as in the earlier examples. Comparison of these results with those obtained earlier for Algorithm 2, figures 10 and 11, shows a slight improvement in accuracy at $t = 0.2$ and a large improvement at $t = 10$. Notice especially, for the Leapfrog algorithm, that the peak at $t = 10$ has not decreased nearly so much, and the shape of the curve is better preserved.

### 4.2 Analysis of the Leapfrog algorithm

We analyze the Leapfrog algorithm almost like we did for Algorithms 1 and 2: we again consider the effect of the algorithm on a single Fourier component, but the computation of $A$ is a little different because the formula for the
[Algorithm 3: Leapfrog]

\[ \phi_{0,J-1,0} = f(x_{0,J-1}) \]

Insert border values for periodic boundary conditions:
\[ \phi_{-1,0} = \phi_{J-1,0}, \quad \phi_{J,0} = \phi_{0,0} \]

First time step:
\[ \phi_{0,J-1,1} = \phi_{0,J-1,0} - \frac{\Delta t}{2} (\phi_{1,J,0} - \phi_{-1,J-2,0}) \]

Main time stepping loop:
for \( n = 1 : N - 1 \)

\[ \phi_{-1,n} = \phi_{J-1,n}, \quad \phi_{J,n} = \phi_{0,n} \]
\[ \phi_{0,J-1,n+1} = \phi_{0,J-1,n} - \Delta t (\phi_{1,J,n} - \phi_{-1,J-2,n}) \]

Figure 19: Leapfrog: Algorithm 3.

Figure 20: The solution computed by the Leapfrog algorithm, marked by "o", and the exact solution, marked by "+", at \( t = 0.2 \); with \( \Delta t = \Delta x = 0.01 \) and \( u = 0.4 \).
Leapfrog algorithm involves three time levels, rather than only the two that were required in Algorithms 1 and 2.

Skipping a few of the initial steps, which should now be familiar to you, we have the following expression for the computed solution:

\[ \phi_{0,J-1,n} = A^n \phi_{0,J-1,0}. \]

Next we determine \( A \) by substituting this solution into equation (25), the Leapfrog equation. This yields

\[ A^{n+1} \phi_{0,J-1,0} = A^{n-1} \phi_{0,J-1,0} - \alpha A^n \phi_{0,J-1,0}(e^{i\omega \Delta x} - e^{-i\omega \Delta x}). \]

Move the terms to the left side, do a little more algebra, and this becomes

\[ A^2 + 2i\alpha A \sin(\omega \Delta x) - 1 = 0. \]

The solutions of this quadratic equation are:

\[ A_+ = -i\alpha \sin(\omega \Delta x) + (1 - \alpha^2 \sin^2(\omega \Delta x))^{1/2}, \quad (27) \]
\[ A_- = -i\alpha \sin(\omega \Delta x) - (1 - \alpha^2 \sin^2(\omega \Delta x))^{1/2}. \quad (28) \]

Therefore we have two solutions of equation (25), the “+” solution

\[ \phi_{0,J-1,n} = A^n_+ \phi_{0,J-1,0}, \]
and the "-" solution
\[ \phi_{0, J-1, n} = A_{-}^{n} \phi_{0, J-1, 0} . \]

Furthermore, since equation (25) is linear, any linear combination of these two solutions is also a solution of this equation. Thus the general solution of equation (25) is
\[ \phi_{0, J-1, n} = c_{1} A_{+}^{n} \phi_{0, J-1, 0} + c_{2} A_{-}^{n} \phi_{0, J-1, 0} , \]
where \( c_{1} \) and \( c_{2} \) are constants determined by the initial conditions. We return to this point later but now let us look closely at the "+" and "-" solutions.

Following the path taken earlier we put \( A_{+} \) and \( A_{-} \) into polar form:
\[ A_{+} = \rho_{+} e^{i \theta_{+}} , \quad A_{-} = \rho_{-} e^{i \theta_{-}} . \]  

Then we determine \( \rho_{+}, \theta_{+}, \rho_{-}, \) and \( \theta_{-} \) from equations (27) and (28). Notice that the square root term in these equations,
\[ (1 - \alpha^{2} \sin^{2}(\omega \Delta x))^{1/2} , \]
can be real or imaginary according to the magnitude of \( \alpha \), therefore we distinguish two cases: \( \alpha \leq 1 \), and \( \alpha > 1 \). If \( \alpha \leq 1 \) it is real for all values of \( \omega \Delta x \); if \( \alpha > 1 \) it is imaginary for some values of \( \omega \Delta x \). We see that this distinction is crucial for stability: the case \( \alpha \leq 1 \) giving a stable algorithm; the case \( \alpha > 1 \) giving an unstable algorithm.

### 4.2.1 The case \( \alpha \leq 1 \)

If \( \alpha \leq 1 \) then with some straightforward algebra
\[ \rho_{+} = 1 , \quad \theta_{+} = - \arctan \left( \frac{\alpha \sin(\omega \Delta x)}{(1 - \alpha^{2} \sin^{2}(\omega \Delta x))^{1/2}} \right) , \]
and
\[ \rho_{-} = 1 , \quad \theta_{-} = \arctan \left( \frac{\alpha \sin(\omega \Delta x)}{(1 - \alpha^{2} \sin^{2}(\omega \Delta x))^{1/2}} \right) + \pi . \]

Since \( \rho_{+} \) and \( \rho_{-} \) are both unity, the solutions corresponding to \( A_{+} \) and \( A_{-} \) are stable and nondissipative.
Following our earlier consideration of $\theta$ in Algorithms 1 and 2, the phase velocity is given by the following equation:

$$u_+^{(\text{computed})} = -\frac{\theta_+}{\omega \Delta t},$$

$$= \arctan \left( \frac{\alpha \sin(\omega \Delta x)}{(1 - \alpha^2 \sin^2(\omega \Delta x))^{1/2}} \right) \frac{1}{\omega \Delta t}.$$  \hspace{1cm} (30)

Figure 22 shows the phase velocity, $u_+^{(\text{computed})}$, as a function of frequency for $\alpha = 0.4$. This should be compared with the corresponding figures, Figures 14 and 16, for Algorithms 1 and 2. Notice that the dispersion in the low frequency waves is comparable to Algorithm 1 and significantly worse than for Algorithm 2.

The consideration of $\theta_-$ is a little different because of the term $\pi$. If we split it off so that we have

$$e^{i \theta_-} = e^{i \pi} e^{i(\theta_- - \pi)} = -e^{i(\theta_- - \pi)}$$  \hspace{1cm} (31)

then the phase velocity for the “-” solution is given by

$$u_-^{(\text{computed})} = -\frac{\theta_- - \pi}{\omega \Delta t}.$$
Advection

\[
\begin{align*}
&= - \arctan \left( \frac{\alpha \sin(\omega \Delta x)}{(1 - \alpha^2 \sin^2(\omega \Delta x))^{1/2}} \right) \frac{1}{\omega \Delta t} \\
&= -u_+^{(\text{computed})}.
\end{align*}
\]

Thus the "-" solution can be seen as a wave moving with the same speed as the "+" solution but in the opposite direction. This wave is somewhat peculiar because it changes sign at every step due to the factor \(e^{i\pi} = -1\) which we had split off: note that from equations (29, 31)

\[A_- = -\rho_- e^{i(\theta_--\pi)}.
\]

If we observe the "-" solution only at every other time step, then it looks like a normal wave travelling in the direction of \(-x\).

Considering the limiting behavior with \(\Delta x \to 0\) and \(\Delta t \to 0\), as before, we find

\[u_+^{(\text{computed})} = u + O((\omega \Delta x)^2).
\]

Thus the "+" solution converges to the exact solution as the grid spacing goes to zero. The "-" solution is an artifact of the numerical method and is not physically meaningful.

In the special case that \(\alpha = 1\), the "+" solution and the exact solution are identical, for we have

\[
\begin{align*}
&u_+^{(\text{computed})} = -\frac{\theta_+}{\omega \Delta t}, \\
&= \arctan \left( \frac{\sin(\omega \Delta x)}{(1 - \sin^2(\omega \Delta x))^{1/2}} \right) \frac{1}{\omega \Delta t}, \\
&= \arctan \left( \frac{\sin(\omega \Delta x)}{\cos(\omega \Delta x)} \right) \frac{1}{\omega \Delta t}, \\
&= \frac{\omega \Delta x}{\omega \Delta t}, \\
&= u.
\end{align*}
\]

The final line in this sequence of equalities may need clarification: recall that \(\alpha = u \Delta t / \Delta x\), consequently \(\alpha = 1\) implies \(u = \Delta x / \Delta t\). Notice that in this case a particle moving with the fluid moves from one grid point to the next in each time step – thus we might have expected something special in this case.
4.2.2 The case $\alpha > 1$

If $\alpha > 1$ then it is possible that $\alpha^2 \sin^2(\omega \Delta x) > 1$ and in this case $A_+$ and $A_-$ are pure imaginary (see equations 27 and 28). In particular we have

$$A_+ = i(-\alpha \sin(\omega \Delta x) + (\alpha^2 \sin^2(\omega \Delta x) - 1)^{1/2}),$$

$$A_- = i(-\alpha \sin(\omega \Delta x) - (\alpha^2 \sin^2(\omega \Delta x) - 1)^{1/2}).$$

hence

$$\rho_+ = \alpha \sin(\omega \Delta x) - (\alpha^2 \sin^2(\omega \Delta x) - 1)^{1/2}, \quad \theta_+ = -\frac{\pi}{2},$$

and

$$\rho_- = \alpha \sin(\omega \Delta x) + (\alpha^2 \sin^2(\omega \Delta x) - 1)^{1/2}, \quad \theta_- = -\frac{\pi}{2}.$$ 

It is evident that $\rho_- > 1$, hence the Leapfrog algorithm is unstable when $\alpha > 1$.

4.2.3 The general solution

We have seen that the numerical solution provided by the Leapfrog algorithm is a linear combination of the “+” solution and the “-” solution:

$$\phi_{0,j-1,n} = c_1 A_+^n \phi_{0,j-1,0} + c_2 A_-^n \phi_{0,j-1,0}. \quad (32)$$

Also, we have seen that if $\alpha \leq 1$ then the “+” solution is a reasonably good approximation to the solution, and the “-” solution is a bad approximation. Therefore $c_2$ must be small relative to $c_1$ if the Leapfrog algorithm is to be accurate. We turn our attention now to determining these two constants.

From equation (32) and the values for $\phi_{0,j-1,0}$ and $\phi_{0,j-1,1}$ we solve the following system of equations for $c_1$ and $c_2$:

$$\phi_{0,j-1,0} = c_1 A_+^0 \phi_{0,j-1,0} + c_2 A_-^0 \phi_{0,j-1,0},$$

$$\phi_{0,j-1,1} = c_1 A_+^1 \phi_{0,j-1,0} + c_2 A_-^1 \phi_{0,j-1,0}.$$

The first of these equations easily reduces to

$$c_1 + c_2 = 1.$$

The second equation, making use of the special first step, becomes

$$\phi_{0,j-1,0} - \frac{\alpha}{2}(\phi_{1,j,0} - \phi_{-1,j-2,0}) = c_1 A_+ \phi_{0,j-1,0} + c_2 A_- \phi_{0,j-1,0}.$$
Then, using a single Fourier component for the initial condition,
\[ \phi_{0,J-1,0} = e^{i\omega x_0;J-1}, \]
the second equation becomes
\[ c_1 A_+ + c_2 A_- = 1 - i\alpha \sin(\omega \Delta x). \]
Substitution of the values for \( A_+ \) and \( A_- \) and then solving the two equations for \( c_1 \) and \( c_2 \) gives
\[
c_1 = \frac{1}{2} \left( 1 + (1 - \alpha^2 \sin^2(\omega \Delta x))^{-1/2} \right),
\]
\[
c_2 = \frac{1}{2} \left( 1 - (1 - \alpha^2 \sin^2(\omega \Delta x))^{-1/2} \right).
\]
Therefore as \( \Delta x \to 0 \), keeping \( \alpha \) constant, \( c_1 \to 1 \) and \( c_2 \to 0 \), and we see that the “+” solution dominates in this limit. More specifically, we have
\[
c_1 = 1 + \left( \frac{\alpha \omega \Delta x}{2} \right)^2 + \mathcal{O}(\omega \Delta x^4),
\]
\[
c_2 = -\left( \frac{\alpha \omega \Delta x}{2} \right)^2 + \mathcal{O}(\omega \Delta x^4).
\]
Thus we see that the presence of high-frequency components not only causes dispersion but also causes the unwanted “-” solution to become a significant part of the solution.

## 4.3 Numerical experiments with the Leapfrog algorithm

Here we illustrate some additional results obtained from using the Leapfrog algorithm with different initial conditions, longer execution times, different velocities and different sets of values for \( \Delta x \) and \( \Delta t \).

### 4.3.1 Sine wave

Here we use
\[ \phi_{0,J-1,0} = \sin(\omega x_0;J-1) \]
for the initial condition. Figures 23 and 24 show results for the velocities $u = 0.4$ and $u = 0.8$, respectively. In each figure the initial condition and the computed solution at $t = 20$ and $t = 40$ are shown. Note that, because of the periodic boundary conditions, the exact solution $\phi(x, t)$ is identical to the initial condition $\phi(x, 0)$ at any time $t$ for which the distance $ut$ is an integer. Therefore the three curves representing the initial condition, the exact solution at $t = 20$, and the exact solution at $t = 40$ should coincide for both $u = 0.4$ and $u = 0.8$. Consequently the separation of the curve for the computed solution from that for the initial condition represents the error in the computed solution.

The effect of dispersion is apparent in these figures: at $t = 20$ the sine wave is lagging behind the initial sine wave and at $t = 40$ it is lagging still further. Reducing the size of $\Delta x$ and $\Delta t$ reduces the lag as we would expect (recall equation (30)). We do not see any change in the amplitude of the wave with time which is also expected since the amplification factor is 1.

Comparison of results for the two different velocities shows a slight decrease in the dispersion for the higher velocity. You can use equation (30) to determine what this decrease should be.

Dispersion increases as the frequency of the sine wave increases (recall figure 22). This effect is illustrated in the next set of results, figures 25 and 26, where the frequency of the sine wave is 3.
Figure 23: The solution computed by the Leapfrog algorithm at $t = 20$ and $t = 40$ with $u = 0.4$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition is $\phi(x, 0) = \sin(4\pi x)$. 
Figure 24: The solution computed by the Leapfrog algorithm at $t = 20$ and $t = 40$ with $u = 0.8$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition is $\phi(x, 0) = \sin(4\pi x)$.
Figure 25: The solution computed by the Leapfrog algorithm at $t = 20$ and $t = 40$ with $u = 0.4$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition is $\phi(x,0) = \sin(6\pi x)$. 

\text{Advection}
Figure 26: The solution computed by the Leapfrog algorithm at $t = 20$ and $t = 40$ with $u = 0.8$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition is $\phi(x, 0) = \sin(6\pi x)$.
4.3.2 Triangular pulse

Now we present a similar set of results for the case when $\phi(x, 0)$ is the triangular pulse, equation (7). They are shown in figures 27 and 28. The same parameter sets as for the sine function are used. As we have seen already, dispersion distorts the triangular shape, though not nearly so much as it did for Algorithms 1 and 2.
Figure 27: The solution computed by the Leapfrog algorithm at $t = 20$ and $t = 40$ with $u = 0.4$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the triangular pulse.
Figure 28: The solution computed by the Leapfrog algorithm at $t = 20$ and $t = 40$ with $u = 0.8$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the triangular pulse.
Advection

4.3.3 Square pulse

The square pulse is zero on the interval [0,1] except on a subinterval in the center where it is equal to 1; specifically,

\[
f(x) = \begin{cases} 
0 & \text{if } 0 \leq x \leq 0.25; \\
1 & \text{if } 0.25 < x < 0.75; \\
0 & \text{if } 0.75 \leq x \leq 1.
\end{cases}
\]

The results displayed in figures 29 and 30 use this square pulse for the initial condition. The first figure uses velocity \( u = 0.4 \), the second uses velocity \( u = 0.8 \).

It is evident that there is a much greater distortion of the square pulse than the triangular pulse. Notice that even when the smaller values of \( \Delta x \) and \( \Delta t \) are used, figures 29b and 30b, the distortion remains rather large. The distortion is caused by dispersion. The reason the dispersion is worse for the square pulse than for the triangular pulse is that the amplitude of the high frequency components of the square pulse are larger than for the triangular pulse, hence the effect of dispersion is more noticeable.

We can make this more explicit by looking at the two sets of Fourier coefficients. We saw earlier, equations (10) and (11), that the Fourier coefficients for the triangular pulse are

\[
c_\nu = \begin{cases} 
1/4 & \text{if } \nu = 0; \\
-4/(\nu \pi)^2 & \text{if } \nu \text{ is odd}; \\
0 & \text{if } \nu \text{ is divisible by } 4; \\
8/(\nu \pi)^2 & \text{otherwise}
\end{cases}, \quad s_\nu = 0.
\]

The Fourier coefficients for the square pulse are

\[
c_\nu = \begin{cases} 
1/2 & \text{if } \nu = 0; \\
-2/\nu \pi & \text{if } \nu \text{ is } 1, 5, 9, \ldots; \\
2/\nu \pi & \text{if } \nu \text{ is } 3, 7, 11, \ldots; \\
0 & \text{if } \nu \text{ is even}
\end{cases}, \quad s_\nu = 0.
\]

The important thing to notice in comparing these two sets of coefficients is that the coefficients for the triangular pulse decrease proportional to \((1/\nu)^2\), while those for the square pulse decrease proportional to \(1/\nu\). Thus high frequencies are much stronger in the square pulse.
Figure 29: The solution computed by the Leapfrog algorithm at $t = 20$ and $t = 40$ with $u = 0.4$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the square pulse.
Figure 30: The solution computed by the Leapfrog algorithm at $t = 20$ and $t = 40$ with $u = 0.8$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the square pulse.
4.3.4 Gaussian pulse

Our final example uses a Gaussian pulse for the initial condition, defined by

\[ f(x) = e^{-250(x-1/2)^2} . \]

The results obtained from using this function as the initial condition are displayed in figures 31 and 32; the first is for velocity \( u = 0.4 \), the second is for velocity \( u = 0.8 \).

Here we can see a significant improvement in the accuracy of the results when \( \Delta x \) and \( \Delta t \) are decreased to 0.005. However, the peak lags behind its correct position, a result of the slower speed of the Fourier components and we see substantial error on the upstream side (left side of the peak) caused by dispersion.
Figure 31: The solution computed by the Leapfrog algorithm at $t = 20$ and $t = 40$ with $u = 0.4$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the Gaussian pulse.
Figure 32: The solution computed by the Leapfrog algorithm at $t = 20$ and $t = 40$ with $u = 0.8$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the Gaussian pulse.
Figure 33: The Trapezoid algorithm is based on finite difference approximations of the partial derivatives $\phi_x$ and $\phi_t$ at $(j, n + 1/2)$ in this figure. The template for the Trapezoid algorithm is indicated by the filled circles.

4.4 Trapezoid algorithm

The Trapezoid algorithm is based on finite difference approximations of the partial derivatives $\phi_x$ and $\phi_t$ at intermediate points in the grid. In particular, to obtain the solution at time $n + 1$ we use approximations of the partial derivatives at $(j, n + 1/2)$, $j = 0, 1, 2, \ldots, J - 1$. The points used in the approximation of $\phi_x$ and $\phi_t$ at $(j, n + 1/2)$ are shown as filled circles in figure 33.

These approximations are

$$
\phi_t(j \Delta x, (n + 1/2) \Delta t) \approx \frac{\phi_{j,n+1} - \phi_{j,n}}{\Delta t},
$$

$$
\phi_x(j \Delta x, (n + 1/2) \Delta t) \approx \frac{1}{2} \left( \frac{\phi_{j+1,n} - \phi_{j-1,n}}{2 \Delta x} + \frac{\phi_{j+1,n+1} - \phi_{j-1,n+1}}{2 \Delta x} \right).
$$

The origin of the first of these should be evident from the earlier discussions. The second is easily seen to be the arithmetic mean of two approximations to $\phi_x$, one at time level $n$, the other at time level $n + 1$. Substitution of these
approximations into the advection equation gives

\[
\frac{\phi_{j,n+1} - \phi_{j,n}}{\Delta t} + \frac{u}{2} \left( \frac{\phi_{j+1,n} - \phi_{j-1,n}}{2\Delta x} + \frac{\phi_{j+1,n+1} - \phi_{j-1,n+1}}{2\Delta x} \right) = 0. \tag{33}
\]

This equation is the basis of the Trapezoid algorithm. Using techniques from our earlier discussion you can verify that this equation differs from the advection equation by an error term that is \(\mathcal{O}((\Delta t)^2)\). Thus it follows that the accuracy of the Trapezoid algorithm is second order, the same as for the Leapfrog algorithm.

This equation involves values of \(\phi\) at only one time level, \(n\), to determine the solution at the next time level, \(n + 1\). In order to see the essence of the Trapezoid algorithm let us put all of the terms involving time level \(n + 1\) on the left side and the terms involving time level \(n\) on the right side, thus

\[-\alpha \phi_{j-1,n+1} + 4\phi_{j,n+1} + \alpha \phi_{j+1,n+1} = \alpha \phi_{j-1,n} + 4\phi_{j,n} - \alpha \phi_{j+1,n}, \tag{34}\]

where \(\alpha = u\Delta t/\Delta x\) as before. Clearly, we cannot use this equation by itself to determine a value of \(\phi\) at time level \(n + 1\) from values of \(\phi\) we already know at time level \(n\). We have to solve a system of equations, obtained by writing equation \((34)\) for \(j = 0, 1, 2, \ldots, J - 1\).

These equations are illustrated with a small example, \(J = 5\). For \(j = 0\) we have

\[-\alpha \phi_{4,n+1} + 4\phi_{0,n+1} + \alpha \phi_{1,n+1} = \alpha \phi_{4,n} + 4\phi_{0,n} - \alpha \phi_{1,n}.
\]

Here the periodic boundary conditions were used to replace \(\phi_{-1,n+1}\) with \(\phi_{4,n+1}\) and \(\phi_{-1,n}\) with \(\phi_{4,n}\). For \(j = 1\) we have

\[-\alpha \phi_{0,n+1} + 4\phi_{1,n+1} + \alpha \phi_{2,n+1} = \alpha \phi_{0,n} + 4\phi_{1,n} - \alpha \phi_{2,n};
\]

and so on. Thus the system of equations we obtain is (in matrix form)

\[
\begin{pmatrix}
4 & \alpha & 0 & 0 & -\alpha \\
-\alpha & 4 & \alpha & 0 & 0 \\
0 & -\alpha & 4 & \alpha & 0 \\
0 & 0 & -\alpha & 4 & \alpha \\
\alpha & 0 & 0 & -\alpha & 4
\end{pmatrix}
\begin{pmatrix}
\phi_{0,n+1} \\
\phi_{1,n+1} \\
\phi_{2,n+1} \\
\phi_{3,n+1} \\
\phi_{4,n+1}
\end{pmatrix}
= 
\begin{pmatrix}
\alpha \phi_{4,n} + 4\phi_{0,n} - \alpha \phi_{1,n} \\
\alpha \phi_{0,n} + 4\phi_{1,n} - \alpha \phi_{2,n} \\
\alpha \phi_{1,n} + 4\phi_{2,n} - \alpha \phi_{3,n} \\
\alpha \phi_{2,n} + 4\phi_{3,n} - \alpha \phi_{4,n} \\
\alpha \phi_{3,n} + 4\phi_{4,n} - \alpha \phi_{0,n}
\end{pmatrix}.
\tag{35}\]
This system must be solved in order to advance the solution one time step. Notice that nothing special needs to be done at the start. The initial conditions are used to evaluate the right-hand side of equation (35) for \( n = 0 \), then the equations can be solved to obtain the solution for \( n = 1 \). This solution can be substituted into the right-hand side and the equations solved again to obtain the solution at \( n = 2 \), and so on.

The right-hand side vector can be expressed as a matrix vector product very similar to the one on the left-hand side, namely

\[
\begin{pmatrix}
\alpha \phi_{4n} + 4\phi_{0,n} - \alpha \phi_{1,n} \\
\alpha \phi_{0,n} + 4\phi_{1,n} - \alpha \phi_{2,n} \\
\alpha \phi_{1,n} + 4\phi_{2,n} - \alpha \phi_{3,n} \\
\alpha \phi_{2,n} + 4\phi_{3,n} - \alpha \phi_{4,n} \\
\alpha \phi_{3,n} + 4\phi_{4,n} - \alpha \phi_{0,n}
\end{pmatrix}
= \begin{pmatrix}
4 - \alpha & 0 & 0 & \alpha \\
\alpha & 4 - \alpha & 0 & 0 \\
0 & \alpha & 4 - \alpha & 0 \\
0 & 0 & \alpha & 4 - \alpha \\
-\alpha & 0 & 0 & \alpha & 4
\end{pmatrix}
\begin{pmatrix}
\phi_{0,n} \\
\phi_{1,n} \\
\phi_{2,n} \\
\phi_{3,n} \\
\phi_{4,n}
\end{pmatrix}.
\]

Notice that this matrix is the same as the coefficient matrix, equation (35), with the sign of \( \alpha \) reversed everywhere.

The form of the system of equations for any \( J \) should be evident from this example. The coefficient matrix is a \( J \times J \) array that is almost tridiagonal, having 4 on the main diagonal, \(-\alpha\) on the lower codiagonal, \( \alpha \) on the upper codiagonal, \( \alpha \) in the lower left corner, \(-\alpha\) in the upper right corner, and zeros elsewhere. The nonzero values in the lower left corner and the upper right corner are a consequence of the periodic boundary conditions. The form of the right-hand side for any \( J \) should also be evident. In the subsequent discussion we use \( M \) to denote the coefficient matrix and \( b_{0,J-1,n} \) to denote the right-hand side, so we write the system to be solved

\[
M \phi_{0,J-1,n+1} = b_{0,J-1,n}.
\]

It is important to notice that \( M \) is constant, a consequence of the fact that we are assuming \( u \) is constant. To verify your understanding of this derivation of the equations to be solved, you can now try deriving them under the assumption that the velocity is not constant, using equation (3).

Thus we arrive at the algorithm shown in figure 34.

For the moment we do not discuss the solution of the system of equations except to make a few observations. Since \( M \) is constant we need to perform an \( LU \) factorization only once, not at every time step. Also, the simple sparse structure of \( M \) makes it unnecessary to store the full matrix.
[Algorithm 4: Trapezoid]

Set initial values:
\[ \phi_{0,J-1,0} = f(x_{0,J-1}) \]

Construct coefficient matrix \( M \):
for \( j = 0 : J - 1 \)
\[ M_{j,j} = 4 \]
for \( j = 0 : J - 2 \)
\[ M_{j,j+1} = \alpha \]
for \( j = 1 : J - 1 \)
\[ M_{j-1,j} = -\alpha \]
\[ M_{0,J-1} = -\alpha \]
\[ M_{J-1,0} = \alpha \]

Main time stepping loop:
for \( n = 1 : N \)
    Insert border values:
    \[ \phi_{-1,n-1} = \phi_{J-1,n-1}; \quad \phi_{J,n-1} = \phi_{0,n-1} \]
    Construct right-hand side vector \( b \):
    for \( j = 0 : J - 1 \)
    \[ b_j = \alpha \phi_{j-1,n-1} + 4\phi_{j,n-1} - \alpha \phi_{j+1,n-1} \]
    Solve \( M\phi_{0,J-1,n} = b \)

Figure 34: Trapezoid: Algorithm 4.
Figure 35: The solution computed by the Trapezoid algorithm, marked by “o”, and the exact solution, marked by “+”, at $t = 0.2$: with $\Delta t = \Delta x = 0.01$ and $u = 0.4$.

Nevertheless, the amount of arithmetic required at each time step and the memory requirements are going to be larger than for any of the preceding algorithms. We will see what we get in return for this extra cost.

Figures 35 and 36 show the computed solution and the exact solution at $t = 0.2$ and $t = 10$, using the triangular pulse again for the initial conditions.

Comparison of these figures with the corresponding ones for the Leapfrog algorithm, figures 20 and 21, show only very small differences. Thus it seems that the cost of the Trapezoid algorithm has not paid a dividend. The next example shows that it has.

We repeat the first example above but with $u = 1.03$, instead of $u = 0.4$. Leapfrog produces the results shown in figure 37. On the other hand Trapezoid produces the far more accurate results shown in figure 38. The large errors in the Leapfrog results should not be surprising because Leapfrog is unstable when $\alpha > 1$ and in this computation $\alpha = 1.03$.

As this example shows, the Trapezoid algorithm allows us to obtain accurate results in the region where the Leapfrog algorithm is unstable. Notice that this implies that we can use the Trapezoid algorithm to reduce the number of time steps in a computation. For example, suppose that the solution at $t = 0.2$ when $u = 0.4$ is computed as before, but with $\Delta x = 0.01$ and
Figure 36: The solution computed by the Trapezoid algorithm, marked by “o”, and the exact solution, marked by “+”, at $t = 10.0$: with $\Delta t = \Delta x = 0.01$ and $u = 0.4$.

Figure 37: The solution computed by the Leapfrog algorithm, marked by “o”, and the exact solution, marked by “+”, at $t = 0.2$: with $\Delta t = \Delta x = 0.01$ and $u = 1.03$. 
Figure 38: The solution computed by the Trapezoid algorithm, marked by “o”, and the exact solution, marked by “+”, at $t = 0.2$: with $\Delta t = \Delta x = 0.01$ and $u = 1.03$.

$\Delta t = 0.04$, making $\alpha = 1.6$. The computational effort is one-fourth of its previous value (we now take 5 steps instead of 20). The Trapezoid algorithm gives the results shown in figure 39 which are nearly the same as those obtained earlier when $\Delta t = 0.01$. The Trapezoid algorithm is about three times slower that Leapfrog, therefore a reduction of the computational effort by a factor of four means that the actual time taken by Trapezoid in this computation is about 3/4 of the time taken by Leapfrog.

### 4.5 Analysis of the Trapezoid algorithm

Since the procedure should now be familiar we leave the details to the reader. Assume the numerical solution has the form

$$\phi_{0,J-1,n} = A^n \phi_{0,J-1,0}.$$ 

Then it follows from equation (33) that

$$A = \frac{2 - i \alpha \sin(\omega \Delta x)}{2 + i \alpha \sin(\omega \Delta x)}.$$ 

Using the polar form

$$A = \rho e^{i \theta},$$
Figure 39: The solution computed by the Trapezoid algorithm, marked by “o”, and the exact solution, marked by “+”, at $t = 0.2$: with $\Delta t = 0.04$, $\Delta x = 0.01$ and $u = 0.4$. Note that in this case $\alpha = 1.6$, a value for which the Leapfrog algorithm is unstable.

we find

$$\rho = 1;$$

$$\theta = - \arctan \left( \frac{\alpha \sin(\omega \Delta x)}{1 - (\alpha^2/4) \sin^2(\omega \Delta x)} \right).$$

Therefore the phase velocity of the computed solution is

$$u(\text{computed}) = \arctan \left( \frac{\alpha \sin(\omega \Delta x)}{1 - (\alpha^2/4) \sin^2(\omega \Delta x)} \right) \frac{1}{\omega \Delta t}.$$ 

Thus the Trapezoid algorithm is stable and not dissipative for all values of $\alpha$, and its dispersion is comparable to the Leapfrog algorithm, equation (30).
4.6 Solution of the linear equations

We turn our attention now to the system of linear equations that must be solved at each iteration, equation (36). Consider the coefficient matrix $M$. Is this matrix well-conditioned? Figure 40 shows the 2-norm condition number as a function of $\alpha$ for $J = 100, 200$. Since $\Delta x = 1/J$, the condition numbers shown apply to $\Delta x = 0.01$ and $\Delta x = 0.005$, values used in our computations. Thus we see that $M$ is well conditioned for any $\alpha$ we might reasonably use, and so we can expect good accuracy in the numerical solution.

A straightforward and efficient way to solve these linear equations is to use $LU$ decomposition and backsolving. For a full matrix the number of operations required to compute $L$ and $U$ is proportional to the cube of the order, but for this matrix it is just proportional to the order (i.e., to $J$). We can see this with a small example, a coefficient matrix of order 5. The form of this matrix is

$$
\begin{pmatrix}
  x & x & 0 & 0 & x \\
  x & x & 0 & 0 \\
  0 & x & x & 0 \\
  0 & 0 & x & x \\
  x & 0 & 0 & x \\
\end{pmatrix},
$$

where $x$ denotes a nonzero element. Assume there is no pivoting. After the
first stage of Gauss elimination, in which we put zeros into the first column below the diagonal, we have the form

\[
\begin{pmatrix}
 x & x & 0 & 0 & x \\
 0 & x & x & 0 & x \\
 0 & x & x & x & 0 \\
 0 & 0 & x & x & x \\
 0 & x & 0 & x & x
\end{pmatrix}.
\]

Notice that two new nonzero elements have been introduced: one in the last row and one in the last column. After the second stage in which zeros are put in the second column below the main diagonal we have the form

\[
\begin{pmatrix}
 x & x & 0 & 0 & x \\
 0 & x & x & 0 & x \\
 0 & 0 & x & x & x \\
 0 & 0 & x & x & x \\
 0 & 0 & x & x & x
\end{pmatrix}.
\]

It should now be evident that at the end of this process we have an upper triangular matrix with the form

\[
\begin{pmatrix}
 x & x & 0 & 0 & x \\
 0 & x & x & 0 & x \\
 0 & 0 & x & x & x \\
 0 & 0 & 0 & x & x \\
 0 & 0 & 0 & 0 & x
\end{pmatrix}.
\]

Thus the final matrix only has nonzero elements on the diagonal, the upper codiagonal and the last column. This is the final form of a coefficient matrix of any order. Also, at every stage the number of arithmetic operations performed is a constant: two divides, four multiplies, four adds (the last stage only requires half of these). So we see that the work required to perform the LU decomposition is proportional to \( J \), and remember that this only needs to be done once if \( u \) is independent of \( t \).

We have assumed that no pivoting is required, now we verify that this is reasonable. Pivoting is done to prevent growth of the matrix elements during the elimination process.\(^1\) However, it is possible to show that there is

\(^1\)The reason for this is that the bound on the error in the solution is proportional to the magnitude of the largest element of the matrix in any stage of the elimination process.
Advection

virtually no growth for these matrix elements when pivoting is ignored. The argument is a little tedious so we skip it, only suggesting how it goes. It is easy to see that after the first stage the diagonal element in the second row has the value $4 + \alpha^2/4$. Generalizing this it can be seen after the $k^{th}$ stage the diagonal element in row $k + 1$, $d_{k+1}$, is given by the simple recurrence relation

$$d_{k+1} = 4 + \left( \frac{\alpha^2}{d_k} \right).$$

From this it follows that no diagonal element exceeds $4 + \alpha^2$, excepting the diagonal element in the lower right corner that grows somewhat. Similar arguments can be applied to the off-diagonal elements.

What about storage? Clearly $3J$ memory cells are sufficient to contain $M$ and they are sufficient to contain $M$ at any stage of the reduction process since each fill-in is matched by a new zero element. Of course keeping track of the elements requires a bit of cleverness. Besides the elements of $M$ we need to save the multipliers, and this requires another $2J - 1$ memory cells. In short, we can handle all of the storage required in a rectangular array of $J$ rows and 5 columns.

Other methods could have been used to solve these equations. The matrix $M$ has a special form: each row differs from its predecessor by a cyclic shift of one position to the right. This is called a Toeplitz matrix. There are special algorithms for solving linear systems of equations in which the coefficient matrix is a Toeplitz matrix. However, they provide no real advantage for this problem. Furthermore, if $u$ depends on $x$, then the Toeplitz nature of $M$ is destroyed.

4.7 Numerical experiments with the Trapezoid algorithm

These experiments are like those conducted with the Leapfrog algorithm; however we take advantage of the fact that we can use values of $\alpha > 1$.

4.7.1 Triangular pulse

In the first group of figures, figure 41 and figure 42, the initial condition is the triangular pulse. Figure 41 is to be compared with the corresponding results
for Leapfrog, figure 27. There are small, not very significant, differences between these two sets of results as we might expect. Figure 42 shows results for similar computations but with the velocity, \( u \), equal to 1.2. While the accuracy deteriorates, it is not too bad when \( \Delta x = \Delta t = 0.005 \). Of course Leapfrog would be totally inaccurate in this case because \( \alpha = 1.2 \).

### 4.7.2 Square pulse

The second group of figures, figure 43 and figure 44, show the results obtained when the initial condition is the square pulse. The results are less accurate than for the triangular pulse because, as already explained, dispersion plays a stronger role. Comparison of figure 43 with corresponding results for the triangular pulse, figure 29, shows again that both algorithms are producing results of about the same accuracy.

### 4.7.3 Gaussian pulse

The last group of figures, figure 45 and figure 46, show results obtained when the initial condition is the Gaussian pulse. Figure 45 can be compared with figure 31 for the Leapfrog algorithm. The results in this group show the same characteristics of those seen in the other groups of figures.

### 4.7.4 Comparing the cost of Leapfrog and Trapezoid algorithms

The one advantage that the Trapezoid algorithm seems to enjoy over Leapfrog is that it can handle problems with larger values of velocity wherein \( \alpha > 1 \). But we can certainly apply the Leapfrog algorithm to problems in which \( u = 1.2 \) by simply reducing the ratio \( \Delta t/\Delta x \) so that \( \alpha < 1 \). For example, if we use the parameter values \( \Delta x = 0.005 \), \( \Delta t = \Delta x \times 2/3 \) and \( u = 1.2 \) we have \( \alpha = 0.8 \) and we can expect to obtain results like those obtained earlier for the Trapezoid algorithm when \( u = 0.8 \). The results are be identical because more iterations, time steps, are required thus producing a greater accumulation of roundoff error, and the values used for \( \phi \) are slightly different since a different grid, \( R_G \), is used. Figure 47 shows results for the Leapfrog algorithm with these parameter values and the triangular pulse for the initial condition. They are very close to those shown in figure 28b for the Trapezoid algorithm with \( \Delta x = \Delta t = 0.005 \) and \( u = 0.8 \), though not identical, and they are also
Figure 41: The computed solution, for the Trapezoid algorithm, at $t = 20$ and $t = 40$ with $u = 0.4$: with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the triangular pulse.
Figure 42: The computed solution, for the Trapezoid algorithm, at $t = 20$ and $t = 40$ with $u = 1.2$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the triangular pulse.
Figure 43: The computed solution, for the Trapezoid algorithm, at $t = 20$ and $t = 40$ with $u = 0.4$: with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the square pulse.
Figure 44: The computed solution, for the Trapezoid algorithm, at $t = 20$ and $t = 40$ with $u = 1.2$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the square pulse.
Figure 45: The computed solution, for the Trapezoid algorithm, at $t = 20$ and $t = 40$ with $u = 0.4$: with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the Gaussian pulse.
Figure 46: The computed solution, for the Trapezoid algorithm, at $t = 20$ and $t = 40$ with $u = 1.2$; with $\Delta x = \Delta t = 0.01$ (a), and $\Delta x = \Delta t = 0.005$ (b). The initial condition for $\phi$ is the Gaussian pulse.
Figure 47: The computed solution, for the Leapfrog algorithm, at $t = 20$ and $t = 40$ with $u = 1.2$, $\Delta x = 0.005$, $\Delta t = 2/3 \Delta x$ (a). The initial condition for $\phi$ is the triangular pulse.

very close to those obtained from the Trapezoid algorithm, figure 42b (note scale is slightly different).

Thus similar results can be obtained from the two algorithms even at larger velocities. What about costs — how does the computation time for one compare with that for the other. The added complexity of the Trapezoid algorithm means a larger computation time, and timing experiments show that the Trapezoid algorithm is about three times slower than the Leapfrog algorithm per time step. Since the total computation time is proportional to the number of time steps, we would have to have a reduction of $\Delta t$ by more than one-third before the Leapfrog algorithm would cost more than the Trapezoid algorithm. Thus if $u > 3$ the Trapezoid algorithm would have the advantage. We could also find the Trapezoid algorithm advantageous when $T$ is very large. Then we may want to increase $\Delta t$ in order to reduce the number of time steps. If this causes $\alpha > 1$, we cannot use the Leapfrog algorithm.

Here we have considered a constant velocity field. In cases where the velocity field is not constant, in time and space, situations can arise where the Leapfrog algorithm could become unstable because of an increase of the velocity in some part of the domain. This could be treated by dynamically
changing $\Delta t$ but this would increase the complexity, hence increasing the
cost of the algorithm; alternatively a value for $\Delta t$ could be chosen that is so
small as to guarantee stability throughout the time and space domain of the
computation and this would also increase the cost of the Leapfrog algorithm.
On the other hand, this concern does not arise for the Trapezoid algorithm; it remains stable.
5 Two-dimensional problems

In these problems $\phi$ is a function of two space variables, $x$ and $y$, and time, $t$; that is, $\phi = \phi(x, y, t)$. Thus $\phi$ now represents the density of a substance on a surface, on a particular level of the atmosphere for example.

The two-dimensional advection equation is

$$\frac{\partial \phi}{\partial t} + \frac{\partial u \phi}{\partial x} + \frac{\partial v \phi}{\partial y} = 0,$$

where $u$ and $v$ are the $x$ and $y$ components of the velocity which may be functions of $x$, $y$, $t$; thus

$$\phi = \phi(x, y, t), \quad u = u(x, y, t), \quad v = v(x, y, t).$$

The equation is sometimes written in the more compact, vector form, as

$$\frac{\partial \phi}{\partial t} + \nabla \cdot \mathbf{c}\phi = 0,$$

where $\mathbf{c}$ is the velocity vector.

The mathematical space of interest, $R$, is three-dimensional: $R = [0, X] \times [0, Y] \times [0, T]$. The grid, $R_\Delta$, is a three-dimensional structure with spacing between points given by $\Delta x$, $\Delta y$, and $\Delta t$. A point in $R_\Delta$ is specified by $(j, k, n)$: $j = 0, 1, 2, \ldots, J$; $k = 0, 1, 2, \ldots, K$; $n = 0, 1, 2, \ldots, N$. It represents the point $(j\Delta x, k\Delta y, n\Delta t)$ in $R$.

It is customary to use the vertical axis in $R$ to represent values of $t$ (time) and the horizontal plane to representing the physical space, with coordinate axes $x$ and $y$, as illustrated in figure 48. Here a single plane of $R_\Delta$ is shown at $t = 0$ ($n = 0$) with $\Delta x = \Delta t = 0.1$ The initial conditions for a computation consist of specifying $\phi$ on this plane.

To display a set of values for $\phi$ at a particular time we use the vertical axis to represent the values of $\phi$. Figure 49 is an illustration.

Periodic boundary conditions are expressed with the equations:

$$\phi(0, y, t) = \phi(X, y, t), \quad y \in [0, Y], \quad t \in [0, T],$$

$$\phi(x, 0, t) = \phi(x, Y, t), \quad x \in [0, X], \quad t \in [0, T].$$
Figure 48: Illustration of the space $R$ with a plane of $R_\phi$ superposed at $t = 0$. Grid spacing is $\Delta x = \Delta y = 0.1$.

Figure 49: Representation of $\phi$ for a Gaussian density centered at $(0.5, 0.5)$.
5.1 Leapfrog and Trapezoid algorithms in two dimensions

It should be unnecessary to repeat the details of the finite difference approximations for these algorithms. The only change is that we must use a finite difference approximation for \( \phi_y \) in addition to one for \( \phi_x \). We also allow \( u \) and \( v \) to be functions of \( x \) and \( y \).

Thus one obtains the following Leapfrog formula for determining the solution at the the point \((j, k, n + 1)\):

\[
\phi_{j,k,n+1} = \phi_{j,k,n-1} + \frac{\Delta t}{\Delta x}(u_{j+1,k}\phi_{j+1,k,n} - u_{j-1,k}\phi_{j-1,k,n}) + \frac{\Delta t}{\Delta y}(v_{j,k+1}\phi_{j,k+1,n} - v_{j,k-1}\phi_{j,k-1,n}).
\]

The computation is very much like the one-dimensional case: it is explicit, and it requires a special step to get started. From this information you should now be able to construct a Leapfrog algorithm for this two-dimensional problem.

The Trapezoid equations are a little more complicated to construct. Following the pattern of our earlier derivation of the Trapezoid algorithm we arrive at the 2-D Trapezoid discretization of the advection equation, the analog of equation 33:

\[
\frac{\phi_{j,k,n+1} - \phi_{j,k,n}}{\Delta t} + \frac{1}{4\Delta x}(u_{j+1,k}\phi_{j+1,k,n} - u_{j-1,k}\phi_{j-1,k,n} + u_{j+1,k}\phi_{j+1,k,n+1} - u_{j-1,k}\phi_{j-1,k,n+1}) + \frac{1}{4\Delta y}(v_{j,k+1}\phi_{j,k+1,n} - v_{j,k-1}\phi_{j,k-1,n} + v_{j,k+1}\phi_{j,k+1,n+1} - v_{j,k-1}\phi_{j,k-1,n+1}) = 0.
\]

If we rearrange the terms so that those at time \( n + 1 \) are on the left and those at time \( n \) are on the right, then we have

\[
-\alpha_{j-1,k}\phi_{j-1,k,n+1} - \beta_{j,k-1}\phi_{j,k-1,n+1} + 4\phi_{j,k,n+1} + \alpha_{j+1,k}\phi_{j+1,k,n+1} + \beta_{j,k+1}\phi_{j,k+1,n+1} = \\
\alpha_{j-1,k}\phi_{j-1,k,n} + \beta_{j,k-1}\phi_{j,k-1,n} + 4\phi_{j,k,n} - \alpha_{j+1,k}\phi_{j+1,k,n} - \beta_{j,k+1}\phi_{j,k+1,n},
\]

(37)
where, for any \((j, k)\) pair,

\[
\alpha_{j,k} = u_{j,k} \frac{\Delta t}{\Delta x}, \quad \beta_{j,k} = v_{j,k} \frac{\Delta t}{\Delta y}.
\]

As in the one-dimensional problem we again must solve a system of linear
equations to advance the solution from time \(n\) to time \(n + 1\). Equation (37)
is one member of that system, which now consists of \(J \times K\) equations.

To put the system into matrix vector form we must establish an ordering
of the equations. Each equation corresponds to a grid point, that is, a \((j, k)\)
pair. We put the equations into the following order:

\[(0, 0), (1, 0), \ldots, (J - 1, 0), (0, 1), (1, 1), \ldots, (J - 1, K - 1)\]

With this ordering the system of equations to be solved at each time step is

\[M \Phi^{(n+1)} = M(\cdot) \Phi^{(n)},\]

where \(M\) and \(M(\cdot)\) are square matrices with \(J \times K\) rows; and \(\Phi^{(n)}\) is a column
vector with \(J \times K\) elements. These quantities are best described with a small
example.

Let \(J = K = 4\), then the matrices are \(16 \times 16\) and the vector has \(16\)
elements. It is easiest if we look at the matrices and vector in block form.
The solution vector, \(\Phi^{(n)}\) is

\[
\Phi^{(n)} = \begin{pmatrix}
\phi_0^{(n)} \\
\phi_1^{(n)} \\
\phi_2^{(n)} \\
\phi_3^{(n)}
\end{pmatrix}, \text{ where } \Phi_i^{(n)} = \begin{pmatrix}
\phi_{0,i,n} \\
\phi_{1,i,n} \\
\phi_{2,i,n} \\
\phi_{3,i,n}
\end{pmatrix}.
\]

Notice that the order of the elements is the order we established above: the
first element is \(\phi_{0,0,n}\), the second is \(\phi_{1,0,n}\) and so forth, the last element being
\(\phi_{3,3,n}\). The coefficient matrix \(M\) is

\[
M = \begin{pmatrix}
A_0 & B_1 & O & -B_3 \\
-B_0 & A_1 & B_2 & O \\
O & -B_1 & A_2 & B_3 \\
B_0 & O & -B_2 & A_3
\end{pmatrix},
\]
Advection

where

\[
A_i = \begin{pmatrix}
4 & \alpha_{1,i} & 0 & -\alpha_{3,i} \\
-\alpha_{0,i} & 4 & \alpha_{2,i} & 0 \\
0 & -\alpha_{1,i} & 4 & \alpha_{3,i} \\
\alpha_{0,i} & 0 & -\alpha_{2,i} & 4
\end{pmatrix},
\]

and

\[
B_i = \begin{pmatrix}
\beta_{0,i} & 0 & 0 & 0 \\
0 & \beta_{1,i} & 0 & 0 \\
0 & 0 & \beta_{2,i} & 0 \\
0 & 0 & 0 & \beta_{3,i}
\end{pmatrix}
\]

and, finally, \( O \) is a \( 4 \times 4 \) matrix of zeros. The matrix \( M^{(-)} \) on the right hand side is

\[
M^{(-)} = \begin{pmatrix}
A^{(-)}_0 & -B_1 & O & B_3 \\
B_0 & A^{(-)}_1 & -B_2 & O \\
O & B_1 & A^{(-)}_2 & -B_3 \\
-B_0 & O & B_2 & A^{(-)}_3
\end{pmatrix},
\]

where

\[
A^{(-)}_i = \begin{pmatrix}
4 & -\alpha_{1,i} & 0 & \alpha_{3,i} \\
-\alpha_{0,i} & 4 & -\alpha_{2,i} & 0 \\
0 & \alpha_{1,i} & 4 & -\alpha_{3,i} \\
-\alpha_{0,i} & 0 & \alpha_{2,i} & 4
\end{pmatrix},
\]

Thus \( M^{(-)} \) is identical to \( M \) except for reversing the sign on every off-diagonal element.

With this information you should be able to construct and analyze algorithms for the two-dimensional advection equation. As a start you might first return to the easy case in which the velocity is constant and repeat some of the one-dimensional experiments described earlier. For this you only need to make \( u \) or \( v \) zero. Next try a velocity directed diagonally across \( R_g \), for example using \( u = 0.4/\sqrt{2} \) and \( v = 0.4/\sqrt{2} \). Finally, you can begin computations in which the velocities are not constant.

The fact that \( \alpha \) and \( \beta \) are variables complicates the analysis, and you will probably find it helpful to assign bounds on these numbers in order to conclude something useful about stability and dispersion. Following ideas in this tutorial, you can study dispersion by following sine waves in the two-dimensional system, observing their velocity as a function of frequency.
6 Conclusion

In an introductory tutorial such as this one it is not an easy matter to decide what to put in and what to leave out. Our decision was to focus on just a few methods and study them in detail with the idea that you would then have the background to study other methods on your own. You will find descriptions of other methods and pointers to the literature in some of the references already given, especially [Haltiner & Williams 80] and [Mesinger & Arakawa 76]. Another source with some results on amplification of amplitude and phase speed for different algorithms is given in a paper by Wen-Yih Sun [Sun 93].

A considerable amount of time was spent on the analysis of the algorithms because we believe the reader should understand not only what an algorithm does but why. Although you can use an algorithm without knowing the “why”, you cannot do a good job of developing new and better algorithms without this knowledge. Furthermore, even the user can benefit because he is often faced with the possibility that his own errors may have caused some peculiar results in a computation. He has an easier time recognizing whether or not this is true if he understands the “why”.

The particular technique we used for the analysis of these algorithms was based on the Fourier series representation of functions. It is a powerful technique and it demonstrates an important application of Fourier series that you might find useful in other contexts. However, while it was useful in this case its use is restricted to linear systems. An analysis of stability of a non-linear partial differential equation requires other methods.

Periodic boundary conditions have been used throughout this tutorial, primarily because they simplify the computations but they also are used in practice. Another boundary condition you might want to explore is, in one-dimension, \( \phi(0, t) = 0 \) for all \( t \). Thus \( \phi \) is held constant and equal to zero at all grid points along the \( t \)-axis. There are some problems here that you have to deal with: note that a larger range of the \( x \)-axis is needed. A comparison of the computations done here with the same computations using this boundary condition would show the effects of periodic boundary conditions.

In meteorology and climatology serious computations require enormous computing power. This means “supercomputers” and implies some form of parallel computation. We have said little about this important subject here but our use of vector notation in many of the formulas indicates, at least,
the easily parallelizable parts of these computations.
References


Advection

