AST560 2025: The Advection-Diffusion Equation

Ammar H. Hakim

February 20, 2025

Contents

1 Properties of the Advection-Diffusion Equation

The advection-diffusion equation is a fundamental equation that describes the transport of a scalar field, $f(\mathbf{x}, t)$ in a given flow field. For scalar diffusion we can write this equation as

$$\frac{\partial f}{\partial t} + \nabla \cdot (\mathbf{u} f) = \nabla \cdot (\alpha \nabla f). \tag{1}$$

Here $\mathbf{u}(\mathbf{x}, t)$ is a *given* time-dependent flow field, and $\alpha(\mathbf{x}, t)$ is a *given* time-dependent diffusion coefficient. If we take the flow and diffusion as given, this is a *linear equation* for the scalar $f(\mathbf{x}, t)$. Despite the apparent simplicity of this equation it is fundamental to understanding the physics of mixing processes in fluids and plasmas (which can be chaotic), and forms a good model and a starting point for more complicated equations. For example, the Vlasov-Maxwell equation and gyrokinetic equations are both advection-diffusion equations in *phase-space* and though nonlinear, can be solved with schemes similar to those we will develop for this linear equation.

Before we develop numerical methods to solve this equation we will derive some important properties of the continuous system. There are several reasons to do this analysis. First, it allows us to design schemes that also mimic these properties in the discrete limit.

Not all continuous properties are inherited by the discrete scheme, and, in fact, it may be possible that some properties may be *harmful* to mimic. Second, as we will see, a careful study of the proofs of the continuous properties give us hints on how to construct proofs for the discrete properties.

We will begin by deriving a *weak-form* of Eq. (1). Let $w(\mathbf{x}, t)$ be a smooth function[1]. Then, multiply Eq. (1) by w and integrate over an arbitrary volume Ω to get

$$\int_\Omega w \frac{\partial f}{\partial t}\, d^3\mathbf{x} + \oint_{\partial\Omega} w(\mathbf{u}f - \alpha\nabla f)\cdot \mathbf{n}\, ds - \int_\Omega \nabla w \cdot (\mathbf{u}f - \alpha\nabla f)\, d^3\mathbf{x} = 0. \tag{2}$$

In this expression $\partial\Omega$ is the surface bounding the volume Ω and $\mathbf{n}$ is the outward unit normal to $\partial\Omega$. We have rearranged the terms and also used integration by parts.

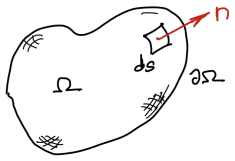

Figure 1: Arbitrary volume Ω, with closed bounding surface $\partial\Omega$ and outward-pointing surface normal $\mathbf{n}$.

This weak-form, in a sense, is more general than the PDE Eq. (1): as the derivatives are no longer on the solution variables, f can be *discontinuous*, and hence the weak-form allows a broader class of solutions than the PDE. Later we will use this weak-form to directly derive discrete schemes in the *discontinuous Galerkin* family of algorithms. For now, we will use the weak-form to prove a couple of important properties of the advection-diffusion equation.

Proposition 1. *The advection-diffusion equation conserves the total amount of scalar quantity. That is*

$$\frac{d}{dt}\int_\Omega f\, d^3\mathbf{x} + \oint_{\partial\Omega} (\mathbf{u}f - \alpha\nabla f)\cdot \mathbf{n}\, ds = 0. \tag{3}$$

Proof. This proof is trivial: just set $w = 1$ in the weak-form Eq. (2). □

This proof shows something that recurs in other conservation laws, and so is important to understand. The proposition says that the amount of "stuff" in Ω is *conserved*, that is, it can only change due to the flow into and out of the volume Ω through its surface $\partial\Omega$. The quantity $(\mathbf{u}f - \alpha\nabla f)$ is called the *flux*. All conservation laws have this structure: some quantity an arbitrary volume can only change due to things flowing in and out of that volume.

Proposition 2. *The advection-diffusion equation monotonically decays the L_2-norm of f if the flow is incompressible, $\nabla\cdot\mathbf{u} = 0$, and $\alpha > 0$. The L_2-norm is conserved if $\nabla\cdot\mathbf{u} = 0$ and $\alpha = 0$.*

[1] A smooth function is one that is continuous and has as many continuous derivatives as we need.

Proof. Choose $w = f$ in the weak-form Eq. (2) to get

$$\frac{d}{dt}\int_\Omega \frac{1}{2}f^2\, d^3\mathbf{x} + \oint_{\partial\Omega} (\mathbf{u}f^2 - \alpha\nabla\frac{1}{2}f^2)\cdot\mathbf{n}\, ds - \int_\Omega \nabla f\cdot(\mathbf{u}f - \alpha\nabla f)\, d^3\mathbf{x} = 0.$$

Now, if $\nabla\cdot\mathbf{u} = 0$ we can write $\nabla f\cdot\mathbf{u}f$ as $\nabla\cdot(\mathbf{u}f^2/2)$ to get

$$\frac{d}{dt}\int_\Omega \frac{1}{2}f^2\, d^3\mathbf{x} + \oint_{\partial\Omega} (\mathbf{u}f^2 - \alpha\nabla\frac{1}{2}f^2)\cdot\mathbf{n}\, ds - \int_\Omega \left(\nabla\cdot\frac{1}{2}\mathbf{u}f^2 - \alpha\|\nabla f\|^2\right) d^3\mathbf{x} = 0.$$

Doing an integration by parts on the first volume term we finally get

$$\frac{d}{dt}\int_\Omega \frac{1}{2}f^2\, d^3\mathbf{x} + \oint_{\partial\Omega} (\mathbf{u}\frac{1}{2}f^2 - \alpha\nabla\frac{1}{2}f^2)\cdot\mathbf{n}\, ds = -\int_\Omega \alpha\|\nabla f\|^2\, d^3\mathbf{x}. \tag{4}$$

If $\alpha > 0$ then this shows that the L_2-norm in Ω decays monotonically, and remains conserved if $\alpha = 0$. □

Note that in this proof it was crucial that the flow is incompressible, $\nabla\cdot\mathbf{u} = 0$. Without this we can't show that the L_2-norm decays. In fact, in a homework problem you will be asked to construct a counter-example that explicitly shows that L_2-norm can *increase or decrease* for compressible flows.

The monotonic decay of the L_2-norm is an important property to ensure in a discrete scheme. In fact, even when $\alpha = 0$, we usually have to ensure that the *discrete scheme decays the L_2-norm*, even though the continuous equation conserves it. This is an example in which the discrete scheme must *not* mimic the continuous properties as otherwise the numerical algorithm is no longer *stable*.

We prove that solution remains positive if it is initially so. This is probably the hardest property to ensure in a discrete scheme.

Proposition 3. *If the solution initially positive everywhere, then it remains so for all times. That is, if $f(\mathbf{x}, 0) > 0$, then $f(\mathbf{x}, t) > 0$, for $t > 0$.*

Proof. Consider a small, cubical volume with sides L in which $f = f_0$ initially. As an extreme case, let the flow be outward on each side and with magnitude U. We can ignore diffusion in this box. Then the weak-form shows that we must have

$$\frac{df}{dt} = -\frac{6U}{L}f. \tag{5}$$

If f_0 and as $U > 0$, we must have that $f \sim e^{-6Ut/L}$. This means that f in the box can only decay towards zero, but never become negative. Any other flow configuration will add f to the box, hence independent of $\mathbf{u}$ the solution will always remain positive. □

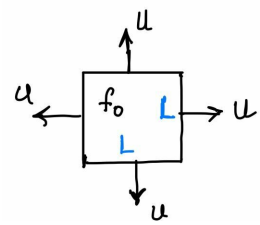

Figure 2: A small, cubical volume with outflow from all sides. If $f_0 > 0$ then then f can only decay towards zero, showing solution always remains positive if it is initially so.

The properties above hold for *arbitrary* volumes, and not just a *specific* volume. Such local properties are far more powerful (and restrictive) than *global* properties. For example, if we were to extend the volume for density conservation to the whole domain, then all one could say is that the total amount of f in the whole domain remains unchanged. However, this would not preclude f disappearing at some point in the domain and instantaneously reappearing somewhere else again. *Local* conservation laws will disallow this. As the volume is arbitrary, the only way f can decrease in this volume is by flux out of the surface.

2 Steps Needed in Constructing a Solver

Now that we have studied a few properties of the equation system we wish to discretize we can switch to constructing the discrete scheme. Notice that we have a *time-dependent* equations with first-order time-derivatives, and first and second order spatial derivatives. Hence, we must

- Choose a *computational grid or mesh* that is appropriate for the problem we wish to solve. For now we will assume simple, rectangular domains and so will use a uniform, rectangular mesh.
- Once we choose the mesh, we use a *discrete approximation* to the spatial derivative terms that appear in the equation. For this, we need to carefully consider where on the mesh to compute the various quantities that appear in our equation, and from this construct a discrete derivative operator of sufficient accuracy. The discrete derivative operators will be different, for example, if we use a curvilinear mesh or a triangular mesh. We will consider such meshes later in the course.
- We then need to decide how to advance the solution in time: we will replace the time derivative with a discrete approximation and *advance* or *march* the solution forward in time using a ODE solver. We will of course need to prescribed initial condition and apply the appropriate *boundary conditions*.

Each of these steps, mesh generation, constructing spatial discrete operators, and choosing a time-stepper are complicated and impact the quality of the numerical solution we will obtain. Once we have made all these choices we can use the *discrete* scheme to prove various properties. Not all properties of the continuous equations are inherited by the discrete scheme. Depending on the set of problems we wish to study, we may want to ensure some key properties of the continuous scheme are inherited by the discrete scheme.

In general, we will also find that there is a tradeoff between accuracy of the scheme and *robustness*. For example, a highly accurate scheme could violate positivity, or will develop spurious high-k modes when the spatial scales of the solution approach the grid spacing. In general, some form of *regularization*, for example, *limiters* or extra diffusion, may be needed to obtain a stable and useful solver.

Of course, implementing the scheme in computer code, specially to create a production solver is highly non-trivial: one needs to have good programming skills and follow good software engineering discipline. The appearance of modern parallel processors and GPUs has made the process even more complicated as, in general, one needs to have a deep understanding of the underlying hardware to obtain best performance from the solver.

Finally, I remark that production solvers applied to "real" physics or engineering problems will invariably encounter issue that will not show up in simple test cases. In fact, if anything *can* go wrong, it *will* go wrong. Eventually. Hence, it is important to have a deep understanding of *both* the physics of the equations *and* the properties of the discrete system. It is seldom fruitful or productive to treat a numerical method as a black-box. The deeper one understand the numerics, the better one can be confident in obtaining useful results from them.

3 Constructing Finite-Difference Formulas

In the advection-diffusion equation we have first-order and second-order derivative operators. We will use *finite-differences* to compute these, however, accounting for some form of *upwinding* for the advection term that we explain below.

Finite-difference (FD) approximations can be systematically constructed by constructing *interpolating polynomials* given values of a function at a set of discrete point. The key approach here is to construct an *interpolating* polynomial in the following way. Let x_i, $i = 1, \ldots, N$ be a set of locations on a 1D grid and $f = f(x_i)$ the corresponding values of f at those points. The first step in constructing FD scheme is to use these to construct a continuous polynomial $f_h(x)$ that satisfies the property that

$$f_h(x_i) = f_i \tag{6}$$

for $i = 1, \ldots, N$. One way to determine this polynomial is to assume a form

$$f_h(x) = a_0 + a_1 x + \ldots + a_{N-1} x^{N-1} \tag{7}$$

where a_j, $j = 0, \ldots, N-1$ are N coefficients. The condition $f_h(x_i) = f_i$ will give us N linear equations to determine the N unknowns a_j, hence determining the polynomial $f_h(x)$.

In reality we do not actually need to construct or solve any linear system. Instead, we can use *Lagrange interpolating* polynomials to construct $f_h(x)$ directly. To illustrate, let $N = 3$. Then the interpolating polynomial is simply

$$f_h(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} f_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} f_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} f_3. \quad (8)$$

Note the construction of this polynomial: when $x = x_1$, for example, the second and third terms vanish, and we get $f_h(x_1) = f_1$, as required. Hence, this quadratic polynomial is the interpolation polynomial we are seeking. Of course, the generalization to more points is obvious.

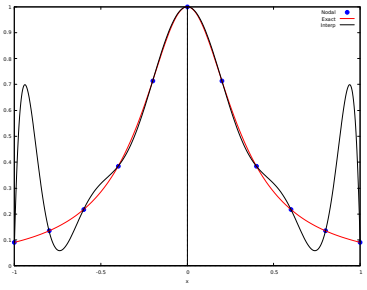

Figure 3: Pathological interpolation of a smooth function on uniformly spaced set of nodes. Black line is the interpolation polynomial that shows severe oscillations, despite the function (red line) being smooth.

Interpolation on uniformly spaced points can often give highly inaccurate results. For example, consider the function

$$f(x) = \frac{1}{1+10x^2} \quad (9)$$

on $x \in [-1, 1]$, This function is smooth but when an interpolating polynomial is constructed using uniformly spaced points it behaves in a pathological manner. One way around this "ringing" is to use a *non-uniformly* spaced sets of points, for example, Legendre or Chebyshev points.

Once we have the interpolation polynomial we can use it to approximate the values and derivatives we need at various places in the algorithm. For example, at some arbitrary point x we can approximate $f(x) = f_h(x)$, or

$$\frac{df(x)}{dx} \approx \frac{df_h(x)}{dx}. \quad (10)$$

Evaluation of the interpolation polynomial and its derivative is often made much easier by use of a compute algebra system (CAS). I highly recommend you use some CAS like Maple, Mathematica, or (my personal choice) Maxima. The use a CAS greatly simplified the various algebraic manipulations needed and also reduces the possibility of errors.

As example, consider we have three cells in a uniform, 1D grid with cell spacing Δx. We will choose to locate the f at *cell-centers*. Then we can use the interpolation formula above to construct a polynomial $f_h(x)$ in the middle cell, $-\Delta x/2 \le x \le \Delta x/2$, assuming, without loss of generality, that $x_0 = 0$. Then, for example, we can compute as

approximation to the second derivative at the point $x_0 = 0$ by computing $d^2 f_h/dx^2$ and evaluating the result at $x = 0$. We get the expected central difference formulas

$$\frac{\partial f_h}{\partial x} = \frac{f_R - f_L}{2\Delta x} \tag{11}$$

$$\frac{\partial^2 f_h}{\partial x^2} = \frac{f_R - 2f_0 + f_L}{\Delta x^2}. \tag{12}$$

This, of course, should not be surprising to you but illustrates the general procedure for computing approximations from the interpolating polynomials.

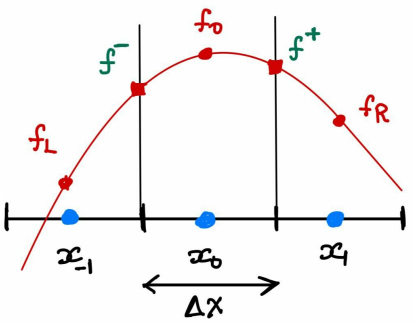

Figure 4: Three cells of a uniform 1D grid. The red curve is a quadratic that fits the cell-center values f_L, f_0 and f_R and is used to construct finite-difference schemes.

As we will see below, we will also need to compute the values f_R and f_L marked in Fig. 4. Again, we simply evaluate the Lagrange interpolating polynomial at $x = \pm\Delta x/2$ to get

$$f^+ = \frac{1}{8}(3f_R + 6f_0 - f_L) \tag{13a}$$

$$f^- = \frac{1}{8}(3f_L + 6f_0 - f_R). \tag{13b}$$

For a smooth $f(x)$ the accuracy of the interpolating polynomial at some point x will depend on the size of the *stencil*, that is, the number of points (cell-centers in this case) that are used in constructing it. To determine the accuracy and convergence of the interpolating polynomial we use a Taylor expansion of $f(x)$ around $x = 0$:

$$f(x) = \sum_{n=0} f^{(n)} \frac{x^n}{n!} \tag{14}$$

where $f^{(n)}$ is the n-th derivative of $f(x)$ evaluated at $x = 0$. We can now use this expansion in the finite-difference expressions and compare with the exact derivatives. For example we can compute

$$f_R = f(\Delta x) = f_0 + \Delta x f^{(1)} + \frac{\Delta x^2}{2} f^{(2)} + \frac{\Delta x^3}{6} f^{(3)} + \ldots \tag{15}$$

$$f_L = f(-\Delta x) = f_0 - \Delta x f^{(1)} + \frac{\Delta x^2}{2} f^{(2)} - \frac{\Delta x^3}{6} f^{(3)} + \ldots. \tag{16}$$

Using this in the finite-difference formula for the first derivative we get

$$\frac{\partial f_h}{\partial x} = \frac{f_R - f_L}{2\Delta x} = f^{(1)} + \frac{\Delta x^2}{6} f^{(3)} + \ldots \tag{17}$$

This shows that with the 3-point stencil, the first derivative computed using the finite-difference formula is *second-order accurate*. In the same way we can compute

$$\frac{\partial^2 f_h}{\partial x^2} = \frac{f_R - 2f_0 + f_L}{\Delta x^2} = f^{(2)} + \frac{\Delta x^2}{12} f^{(4)} + \dots \tag{18}$$

again showing that the second derivative is also *second-order accurate*.

We can also compute the accuracy of the finite-difference formulas for computing the edge values $f^{\pm}$. Using the Taylor expansions in Eq. (13) we get

$$f^{\pm} - f(\pm\Delta x/2) = \pm\frac{\Delta x^3}{16} f^{(3)} + \dots \tag{19}$$

showing that the edge values are computed to *third-order* accuracy.

The validity of the above Taylor series analysis, of course, depends crucially on the fact that the solution $f(x)$ is sufficiently smooth that the needed derivatives exist. This may not always be true, for example, in the presence of shocks or sharp gradients in the solution. It is also clear that the degree of smoothness we need will depend on the stencil size (and the order of the derivative we are computing): hence, wider stencils (more accurate derivatives) will be increasingly *less* robust, with potentially large errors in regions of sharp gradients. In general, this means that unless great care is taken, high order methods (third or higher order) are far *less robust* than lower order methods (second-order). This is one of the key reasons that many commercial computational physics packages, especially for fluid mechanics[2], typically use low-order schemes, sometimes even effectively just first-order schemes.

Besides Taylor series analysis, we can also do a Fourier analysis to understand how a single mode is represented by the finite-difference formula for the derivative operators. To do this we will take a single Fourier mode

$$f(x) = e^{ikx} \tag{20}$$

and substitute this in the finite-difference formula. For, example, for the first derivative we have

$$\frac{\partial f_h}{\partial x} = i\bar{k} = \frac{e^{ik\Delta x} - e^{-ik\Delta x}}{2\Delta x}. \tag{21}$$

[2]Solid mechanics and heat transfer problems can be solved with higher order methods, as the solutions are typically much smoother that fluid problems. In the latter, specially in invicid or low viscosity fluids, the flow tends to cascade to shorter wavelengths (high-k) modes.

Note that we have used a *modified wave-number* $\bar{k}$ on the left as this is what the scheme will actually see, rather than the exact wave-number k. In general, the modified wave-number will have both a real and an imaginary part. In this specific example, as the stencil is symmetric $\bar{k}$ only has a real part. We easily see that this is

$$\bar{k}_R \Delta x = \sin(k\Delta x). \tag{22}$$

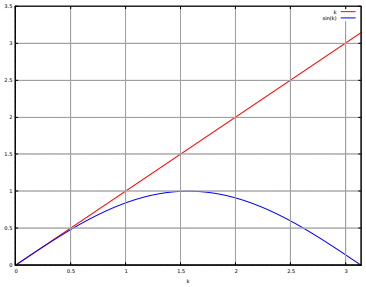

Figure 5: Blue: Numerical wave-number plotted as a function of actual wave-number for central difference scheme for the first derivative. Significant dispersion and aliasing is seen.

The figure shows the numerical wave-number $\bar{k}_R$ plotted against the actual wave-number. Notice there is strong deviation from the red line for even modest waves. This figure shows strong *aliasing* of modes: each long wavelength mode is *aliased* to a corresponding short wavelength mode. Further, this figure also shows that if we were to use the central-difference formula for advection, modes with different wave-numbers would propagate with different phase- and group velocities, causing *dispersion* of waves.