10 Conservation Laws and Heat Flow

Many physical phenomena have a conservation law associated with them. For instance, matter, energy, and momentum are all conserved quantities. The *fundamental conservation law* states that the rate of change of the total quantity in the system is equal to the rate that the quantity enters the system plus the rate at which the quantity is produced by sources inside the system. While this is a *global* property, we can use it to obtain a *local* differential equation that the concentration of the quantity must obey everywhere in the system. Because of this, conservation laws are very important in modeling a wide variety of phenomena.

Derivation of the Conservation equation in multiple dimensions

Suppose Ω is a region in $\mathbb{R}^n$, and $V \subset \Omega$ is bounded with a reasonably well-behaved boundary ∂V. Let $u(\vec{x}, t)$ represent the density (concentration) of some quantity throughout Ω. Let $\vec{n}(x)$ represent the normal direction to V at $x \in \partial V$, and let $\vec{J}(\vec{x}, t)$ be the flux vector for the quantity, so that $\vec{J}(\vec{x}, t) \cdot \vec{n}(x)\, dA$ represents the rate at which the quantity leaves V by crossing a boundary element with area dA. Note that the total amount of the quantity in V is

$$\int_V u(\vec{x}, t)\, dt,$$

and the rate at which the quantity enters V is

$$-\int_{\partial V} \vec{J}(\vec{x}, t) \cdot \vec{n}(x)\, dA.$$

We let the source term be given by $g(\vec{x}, t, u)$; we may interpret this to mean that the rate at which the quantity is produced in V is

$$\int_V g(\vec{x}, t, u)\, dt.$$

Then the integral form of the conservation law for u is expressed as

$$\frac{d}{dt}\int_V u(\vec{x}, t)\, d\vec{x} = -\int_{\partial V} \vec{J} \cdot \vec{n}\, dA + \int_V g(\vec{x}, t, u)\, d\vec{x}.$$

If u and J are sufficiently smooth functions, then we have

$$\frac{d}{dt}\int_V u\, d\vec{x} = \int_V u_t\, d\vec{x},$$

and

$$\int_{\partial V} \vec{J} \cdot \vec{n}\, dA = \int_V \nabla \cdot \vec{J}\, d\vec{x}.$$

Putting these together yields

$$\int_V u(\vec{x}, t)\, d\vec{x} = \int_V \left(-\nabla \cdot \vec{J} + g(\vec{x}, t, u) \right)\, d\vec{x}$$

Since this holds for all nice subsets $V \subset \Omega$ with V arbitrarily small, the integrands must be equal everywhere, and we obtain the differential form of the conservation law for u:

$$u_t + \nabla \cdot \vec{J} = g(\vec{x}, t, u),$$

where ∇ is the gradient operator and $\nabla \cdot \vec{J} = \frac{\partial J_1}{\partial x_1} + \cdots + \frac{\partial J_n}{\partial x_n}$

Constitutive Relations

So far, our conservation law consists of 2 unknowns (u and J) but only 1 equation. To this equation we need to add other equations, called *constitutive relations*, which are used to fully determine the system.

For example, suppose we wish to model the flow of heat. Since heat flows from warmer regions to colder regions, and the rate of heat flow depends on the difference in temperature between regions, we usually assume that the flux vector $\vec{J}$ is given by

$$\vec{J}(x, t) = -\nu \nabla u(x, t),$$

where ν is called the diffusion constant and $\nabla u(x, t) = [\partial_{x_1} u, \ldots, \partial_{x_n} u]^T$. This constitutive relation is called Fick's law, and is the basic model for any diffusive process. Substituting into the conservation law we obtain

$$u_t - \nu \Delta u(x, t) = g(\vec{x}, t, u)$$

where Δ is the Laplacian operator:

$$\Delta u(x, t) = \frac{\partial^2 u}{\partial x_1^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2}.$$

The function g represents heat sources and sinks within the region.

Numerically modeling heat flow

Consider the heat flow equation in one dimension together with an appropriate initial condition $u(x, 0) = f(x)$, homogeneous Dirichlet boundary conditions, and $g(x, t, u) = 0$:

$$\begin{aligned} &u_t = \nu u_{xx}, \quad x \in [a, b], \quad t \in [0, T], \\ &u(a, t) = 0, \quad u(b, t) = 0, \\ &u(x, 0) = f(x). \end{aligned}$$

We will create an approximation U_i^j to $u(x_i, t_j)$ on the grid $x_i = a + hi$, $t_j = kj$, where h and k are small changes in x and t respectively and i and j are indices; so, U_i^j denotes the approximate value of u at the i-th grid point and the j-th time step.

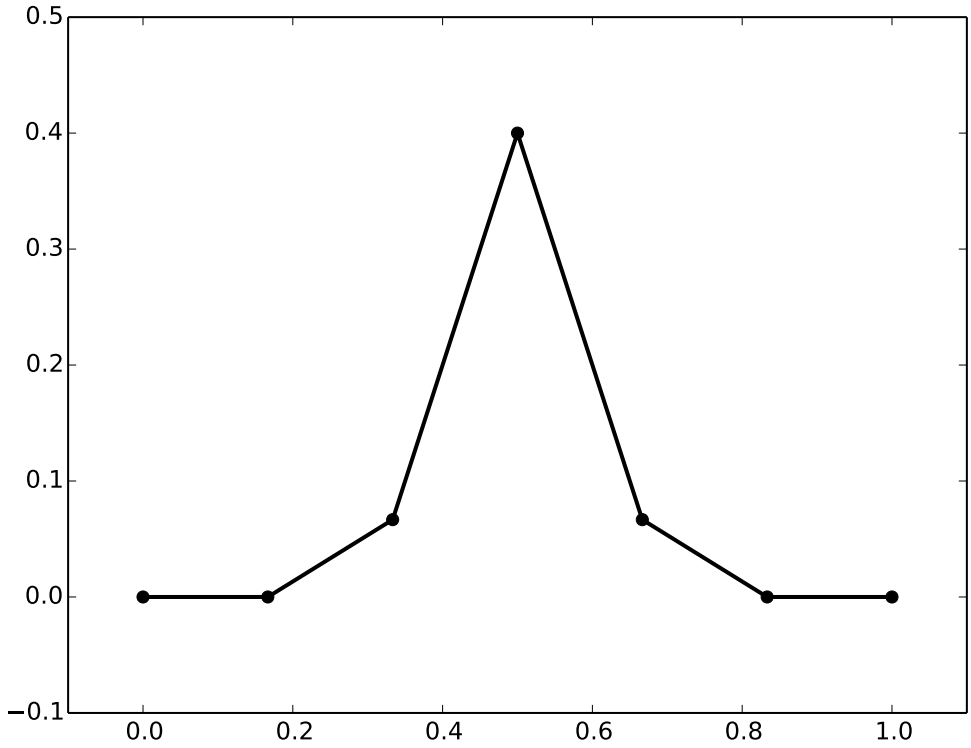

Figure 10.1: The graph of U^0, the approximation to the solution $u(x, t = 0)$ for Problem 1.

As before, we will use the finite difference method to create this approximation. Recall that by using Taylor's theorem, we have the first-order forward difference approximation

$$u_t(x,t) = \frac{u(x,t+k) - u(x,t)}{k} + \mathcal{O}(k).$$

and the second-order centered difference approximation

$$u_{xx}(x_i,t_j) = \frac{u(x_i+h,t_j) - 2u(x_i,t_j) - u(x_i-h,t_j)}{h^2} + \mathcal{O}(h^2).$$

Applying these difference approximations give us the $\mathcal{O}(h^2 + k)$ explicit method

$$\begin{aligned} \frac{U_i^{j+1} - U_i^j}{k} &= \nu \frac{U_{i+1}^j - 2U_i^j + U_{i-1}^j}{h^2}, \\ U_i^{j+1} &= U_i^j + \frac{\nu k}{h^2}(U_{i+1}^j - 2U_i^j + U_{i-1}^j). \end{aligned} \tag{10.1}$$

This method can be written in matrix form as

$$U^{j+1} = AU^j,$$

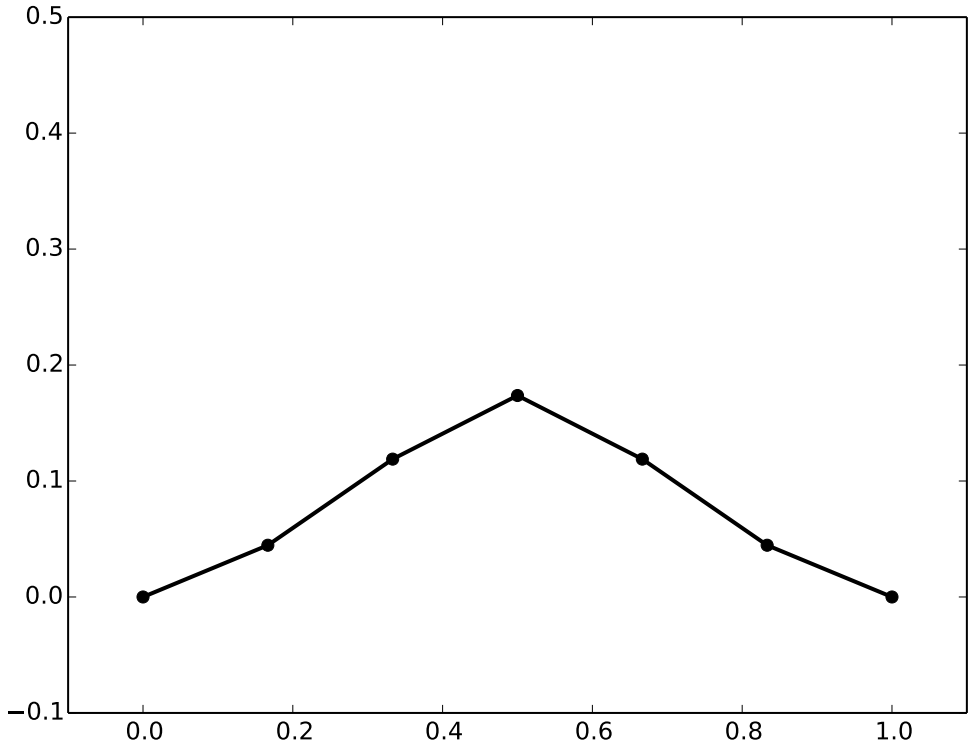

Figure 10.2: The graph of U^4, the approximation to the solution $u(x, t = 0.4)$ for Problem 1.

where A is the tridiagonal matrix given by

$$A = \begin{bmatrix} 1 & 0 & & & \\ \lambda & 1-2\lambda & \lambda & & \\ & \ddots & \ddots & \ddots & \\ & & \lambda & 1-2\lambda & \lambda \\ & & & 0 & 1 \end{bmatrix},$$

$\lambda = \nu k/h^2$, and U^j represents the approximation at time t_j. We can initialize this method using the initial condition given in our problem, which tells us that $U_i^0 = f(x_i)$.

NOTE

Note that the matrix representing the finite difference scheme is very *sparse*, which is typical of finite difference schemes. While represeting finite difference schemes with matrices can be an effective method, especially for implicit schemes, it is very important that a sparse matrix format is used. Otherwise, performance will be dramaticaly negatively impacted. In Python, since looping is slow, the best alternative is to vectorize the difference scheme. This approach can in fact be even better than using matrices for explicit schemes, such as the one we are using here, as it avoids needing to store the matrix in memory.

To account for our constant boundary conditions using this differencing scheme, simply set the boundary points to the appropriate values in the initial conditions, then avoid modifying them as you update for each time step. Note that the first and last rows of the matrix representation of the differencing scheme are the same as the first and last rows of the identity matrix. This has the effect of keeping the boundary points the same as in the previous step, and thus the same as in the initial condition.

Problem 1. Consider the initial/boundary value problem

$$\begin{aligned} &u_t = 0.05u_{xx}, \quad x \in [0,1], \quad t \in [0,1] \\ &u(0,t) = 0, \quad u(1,t) = 0, \\ &u(x,0) = 2\max\{0.2 - |x - 0.5|, 0\}. \end{aligned} \tag{10.2}$$

Approximate the solution $u(x,t)$ by taking 6 subintervals in the x dimension and 10 subintervals in time. Plot the solution at the times $t = 0, t = 0.4$, and $t = 1$. The graphs for U^0 and U^4 are given in Figures 10.1 and 10.2.

Problem 2. Solve the initial/boundary value problem

$$\begin{aligned} &u_t = u_{xx}, \quad x \in [-12,12], \quad t \in [0,1], \\ &u(-12,t) = 0, \quad u(12,t) = 0, \\ &u(x,0) = \max\{1 - x^2, 0\} \end{aligned} \tag{10.3}$$

using the first order explicit method 10.1. Use 140 subintervals in the x dimension and 70 subintervals in time. The initial and final states are shown in Figure 10.3. Animate your results.

Explicit methods usually have a stability condition, called a CFL condition (for Courant-Friedrichs-Lewy). For method 10.1 the CFL condition that must be satisfied is that

$$\lambda = \frac{\nu k}{h^2} \leq \frac{1}{2}.$$

Repeat your computations using 140 subintervals in the x dimension and 66 subintervals in time. Animate the results. For these values, the CFL condition is broken; you should be able to clearly see the result of this instability in the approximation U^{66}.

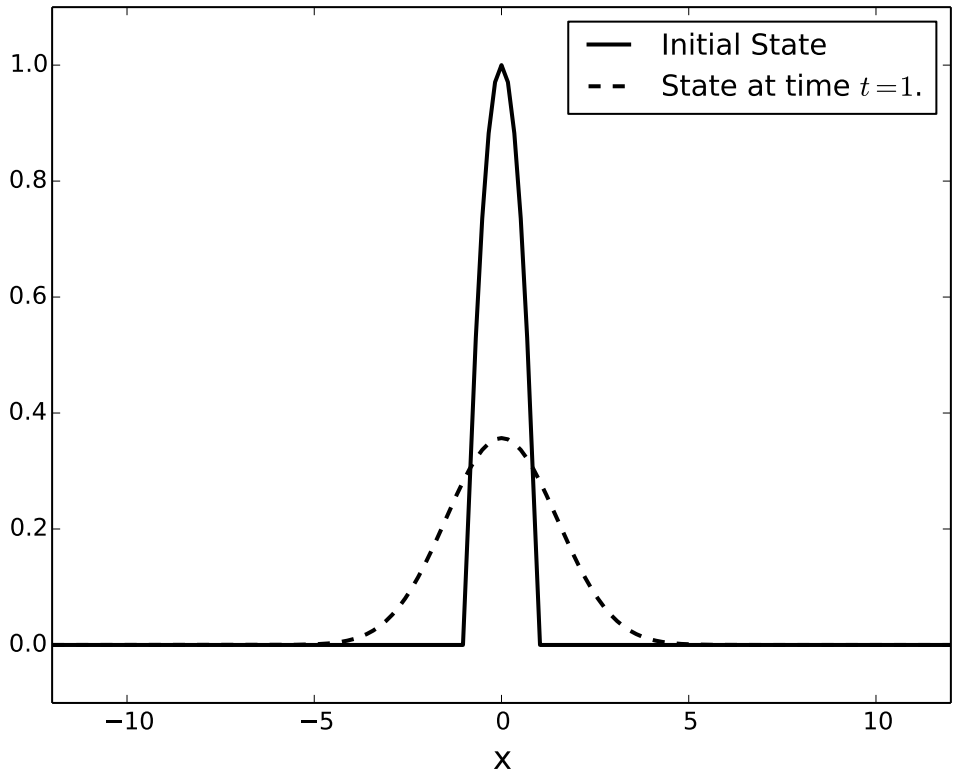

Figure 10.3: The initial and final states for equation Problem 2.

Implicit methods often have better stability properties than explicit methods. The Crank-Nicolson method, for example, is unconditionally stable and has order $\mathcal{O}(h^2 + k^2)$. To derive the Crank-Nicolson method, we use the following approximations:

$$\begin{aligned} u_t(x_i, t_{j+1/2}) &= \frac{u(x_i, t_{j+1}) - u(x_i, t_j)}{k} + \mathcal{O}(k^2), \\ u_{xx}(x_i, t_{j+1/2}) &= \frac{u_{xx}(x_i, t_{j+1}) + u_{xx}(x_i, t_j)}{2} + \mathcal{O}(k^2). \end{aligned}$$

The first equation is a finite difference approximation for u_t, and the second is a midpoint approximation applied to u_{xx}. These approximations give the relation

$$\begin{aligned} \frac{U_i^{j+1} - U_i^j}{k} &= \frac{1}{2}\left(\frac{U_{i+1}^j - 2U_i^j + U_{i-1}^j}{h^2} + \frac{U_{i+1}^{j+1} - 2U_i^{j+1} + U_{i-1}^{j+1}}{h^2}\right), \\ U_i^{j+1} &= U_i^j + \frac{k}{2h^2}\left(U_{i+1}^j - 2U_i^j + U_{i-1}^j + U_{i+1}^{j+1} - 2U_i^{j+1} + U_{i-1}^{j+1}\right). \end{aligned} \tag{10.4}$$

This method can be written in matrix form as

$$BU^{j+1} = AU^j,$$

where A and B are tridiagonal matrices given by

$$B = \begin{bmatrix} 1 & 0 & & & \\ -\lambda & 1+2\lambda & -\lambda & & \\ & \ddots & \ddots & \ddots & \\ & & -\lambda & 1+2\lambda & -\lambda \\ & & & 0 & 1 \end{bmatrix},$$

$$A = \begin{bmatrix} 1 & 0 & & & \\ \lambda & 1-2\lambda & \lambda & & \\ & \ddots & \ddots & \ddots & \\ & & \lambda & 1-2\lambda & \lambda \\ & & & 0 & 1 \end{bmatrix},$$

where $\lambda = \nu k/(2h^2)$, and U^j represents the approximation at time t_j. Note that here we have defined λ differently than we did before!

How do we know if a numerical approximation is reasonable? One way to determine this is to compute solutions for various step sizes h and see if the solutions are converging to something, which we hope to be the true solution. To be more specific, suppose our finite difference method is $\mathcal{O}(h^p)$ accurate. This means that the error $E(h) \approx Ch^p$ for some constant C as $h \to 0$ (that is, for $h > 0$ small enough).

So, we will compute the approximation y_k for each stepsize h_k, $h_1 > h_2 > \ldots > h_m$. We will think of y_m as the true solution. Then the error of the approximation for stepsize $h_k, k < m$, is

$$\begin{aligned} E(h_k) &= \max(|y_k - y_m|) \approx Ch_k^p, \\ \log(E(h_k)) &= \log(C) + p\log(h_k). \end{aligned}$$

Thus on a log-log plot of $E(h)$ vs. h, these values should be on a straight line with slope p when h is small enough to start getting convergence.

Problem 3. Using the Crank Nicolson method, numerically approximate the solution $u(x,t)$ of the problem

$$\begin{aligned} &u_t = u_{xx}, \quad x \in [-12, 12], \quad t \in [0, 1], \\ &u(-12, t) = 0, \quad u(12, t) = 0, \\ &u(x, 0) = \max\{1 - x^2, 0\}. \end{aligned} \tag{10.5}$$

Note that this is an implicit linear scheme; hence, the most efficient way to find U^{j+1} is to create the matrix B as a sparse matrix and use `scipy.sparse.linalg.spsolve`.

Demonstrate that the numerical approximation at $t = 1$ converges. Do this by computing U at $t = 1$ using $20, 40, 80, 160, 320$, and 640 steps. Use the same number of steps in both time and space. Reproduce the loglog plot shown in Figure 10.4. The slope of the line there shows the order of convergence.

To measure the error, use the solution with the smallest h (largest number of intervals) as if it were the exact solution, then sample each solution only at the x-values that are represented in the solution with the largest h (smallest number of intervals). Use the ∞-norm on the arrays of values at those points to measure the error.

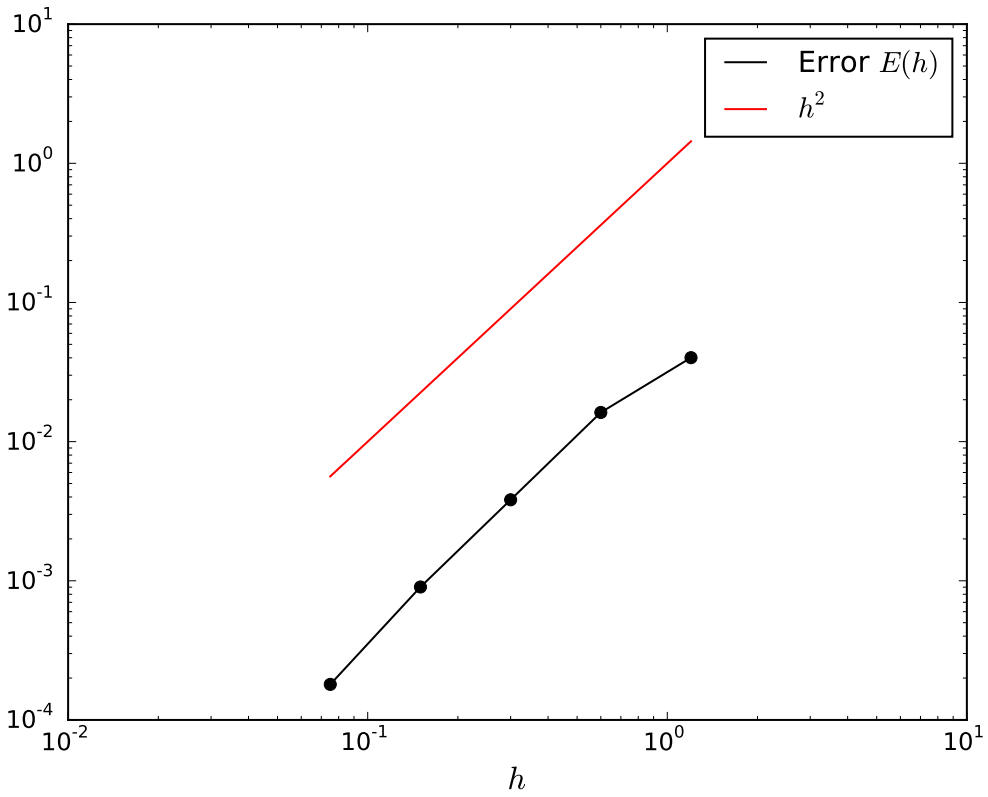

Figure 10.4: $E(h)$ represents the (approximate) maximum error in the numerical solution U to Problem 3 at time $t = 1$, using a stepsize of h.

Notice that, since the Crank-Nicolson method is unconditionally stable, there is no CFL condition, and we can safely use the same number of intervals in time and space.