

Chapter 3

Timestepping Algorithms and the Enthalpy Method

This chapter will expand on the previous description of finite difference discretization of time in the heat equation by providing a framework for implicit and explicit time stepping algorithms. It will then discuss the enthalpy method for tracking phase boundaries e.g. during melting and solidification.

3.1 Introduction

Rodolfo Rosales began this series by discussing conservation laws, with various examples including solute and thermal diffusion. The approach was to treat various phenomena in terms of fields in space and time: a concentration field, a temperature field, etc., with partial differential equations describing the changes in those fields over time.

This section introduces a numerical method called Finite Differences for approximately solving the partial differential equations to give an estimate for the fields themselves. The heat equation is used as an example, with multiple different schemes discussed for time integration (of which explicit time stepping is the only one you will be required to know). This simple but limited approach serves to introduce several aspects of numerical solution of PDEs, from stability of integration methods, to linearization of the equation system into a matrix equation; these will inform your understanding of more powerful but complex methods later.

3.2 Finite Differences and the Energy Equation

The laws of thermodynamics give rise to a general equation for heat conduction given in section 2.4.3 equation 4.7:

$$\rho c \frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial x^2} + s. \quad (2.1)$$

There are many analytical solutions to this equation for various initial and boundary conditions; if generation is zero then Fourier series provide a solution for any initial condition; Green's function integrals solve the steady-state equation straightforwardly for any generation expression (and the time-dependent equation somewhat less straightforwardly). But let's face it, these things are a pain,

it's a lot easier to throw the equations at a computer and let the machine do the work. Furthermore, for complex boundary conditions and/or geometries, one is not guaranteed to obtain such a solution at all; Green's functions often don't integrate well, either analytically or numerically.

Into this need steps the simple computational method of finite differences. To use finite differences, we start by discretizing space and time into a finite number of points, as illustrated in figure 2.1. In one dimension, we can call the spatial points x_i , and the time values t_n . For simplicity, here we will consider only uniform discretization, where we define $\Delta x = x_1 - x_0 = x_{i+1} - x_i$ for all i , and $\Delta t = t_{n+1} - t_n$ for all n . The temperature at position x_i and time t_n can be written $T_{i,n}$.

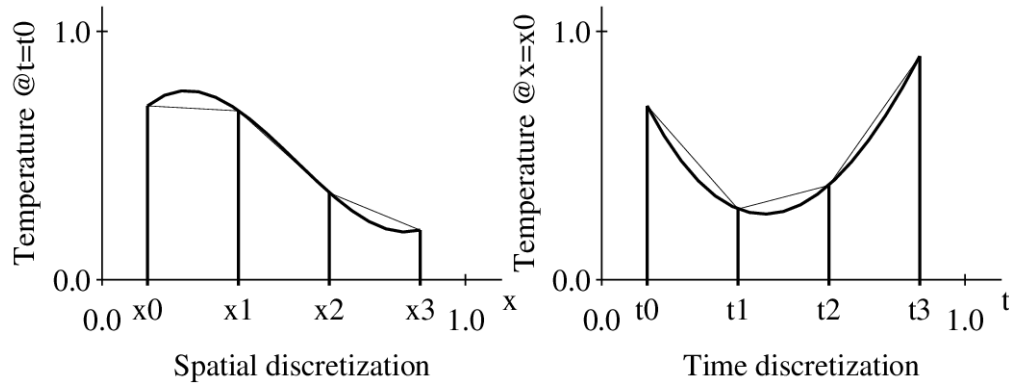


Figure 2.1: Finite difference discretization of space and time. The thin line segments represent the finite difference approximation of the function.

We can then estimate the temperature derivatives as follows:

$$\left. \frac{\partial T}{\partial t} \right|_{x_i, t_{n+1/2}} \simeq \frac{T_{i,n+1} - T_{i,n}}{\Delta t}, \quad (2.2)$$

$$\left. \frac{\partial T}{\partial x} \right|_{x_{i+1/2}, t_n} \simeq \frac{T_{i+1,n} - T_{i,n}}{\Delta x} \quad (2.3)$$

The second derivative $\partial^2 T / \partial x^2$ can then be approximated as a derivative of derivatives:

$$\left. \frac{\partial^2 T}{\partial x^2} \right|_{x_i, t_n} \simeq \frac{\left. \frac{\partial T}{\partial x} \right|_{x_{i+1/2}, t_n} - \left. \frac{\partial T}{\partial x} \right|_{x_{i-1/2}, t_n}}{\Delta x} \simeq \frac{T_{i-1,n} - 2T_{i,n} + T_{i+1,n}}{\Delta x^2} \quad (2.4)$$

3.2.1 Explicit time stepping

These derivative approximations can then be used to estimate solutions to the heat equation (equation 2.1), turning that differential equation into a *difference equation*:

$$\rho c \frac{T_{i,n+1} - T_{i,n}}{\Delta t} = k \frac{T_{i-1,n} - 2T_{i,n} + T_{i+1,n}}{\Delta x^2} + s. \quad (2.5)$$

We can simplify this slightly by recalling the definitions of the thermal diffusivity $\nu = k/\rho c$ and normalized source $f = s/\rho c$ (section 2.4.3, page 15), and defining the *mesh Fourier number* Fo_M as:

$$\text{Fo}_M = \frac{k\Delta t}{\rho c \Delta x^2} = \frac{\nu \Delta t}{\Delta x^2} \quad (2.6)$$

(recall ν is the thermal diffusivity $k/\rho c$), so a rearranged equation 2.5 becomes:

$$T_{i,n+1} = T_{i,n} + \Delta t \left[\nu \frac{T_{i-1,n} - 2T_{i,n} + T_{i+1,n}}{\Delta x^2} + \frac{s}{\rho c} \right] = T_{i,n} + \text{Fo}_M (T_{i-1,n} - 2T_{i,n} + T_{i+1,n}) + f \Delta t \quad (2.7)$$

This gives us a nice algorithm for computing the temperatures at the next timestep from those of the previous timestep and those on the boundaries. This algorithm is called *explicit timestepping*, or the *Forward Euler* timestepping algorithm. It is so straightforward that we can even do it in a simple spreadsheet. But its key drawback is that for large time step sizes, it is unstable, as discussed below.

Explicit timestepping stability criterion

We can regroup the terms on the right side of equation 2.7 to express the temperature in the new timestep as follows:

$$T_{i,n+1} = T_{i,n}(1 - 2\text{Fo}_M) + 2\text{Fo}_M \frac{T_{i-1,n} + T_{i+1,n}}{2} + f \Delta t. \quad (2.8)$$

Neglecting the last term for generation, this is effectively a weighted average between $T_{i,n}$ and the mean of its two neighbors: if $\text{Fo}_M = 0$, then $T_{i,n+1} = T_{i,n}$ (goes nowhere); if $\text{Fo}_M = \frac{1}{2}$, then $T_{i,n+1} = \frac{1}{2}(T_{i-1,n} + T_{i+1,n})$ (mean of the neighbors); if Fo_M is between 0 and $\frac{1}{2}$ then $T_{i,n+1}$ will be between these two. But if $\text{Fo}_M > \frac{1}{2}$, then the new temperature goes beyond the mean of the neighbors. This makes the solution unstable, as oscillations in the solution will grow geometrically with each timestep, as shown in figure 2.2.

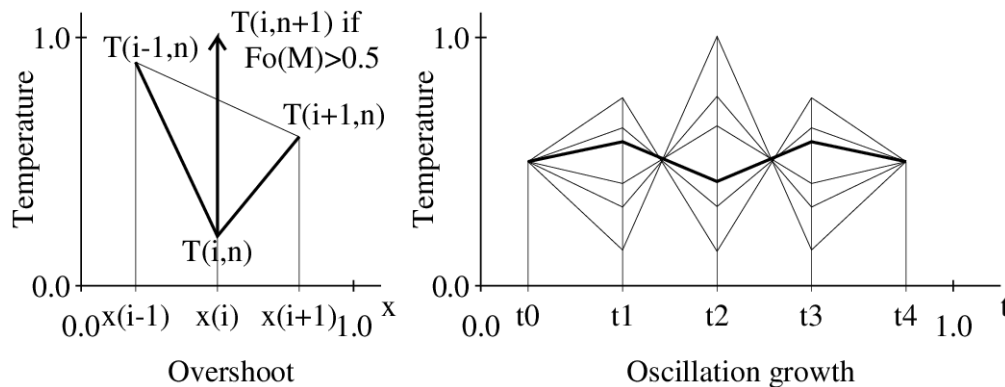


Figure 2.2: “Overshoot” of the average neighboring temperature for $\text{Fo}_M > 0.5$; growth of an oscillation for $\text{Fo}_M = 0.7$ with T_0 and T_4 fixed as boundary conditions (the dark curve is the initial condition).

This result gives the explicit timestepping stability criterion as $\text{Fo}_M \leq \frac{1}{2}$, and restricts the choice of Δx and Δt . Written in terms of Δt , this criterion is:

$$\Delta t \leq \frac{\Delta x^2}{2\nu}. \quad (2.9)$$

Note that in two dimensions with a square grid, there are four neighbors, and the criterion becomes $\text{Fo}_M \leq \frac{1}{4}$; in three dimensions, $\text{Fo}_M \leq \frac{1}{6}$. Equation 2.9 implies that if one makes the spatial discretization twice as fine (cutting Δx in half), then the timestep must be reduced by a factor of four, requiring eight times the computational work to simulate the same amount of total time.

3.2.2 Implicit Timestepping

Explicit timestepping is a simple algorithm for doing timestepping. Unfortunately, the explicit stability criterion places a very strict limit on the timestep size, making computation very expensive for any decent mesh spacing, but there are methods which can use much much bigger timesteps.

First, explicit timestepping extrapolates the spatial derivatives at the present time forward, resulting in an instability. So why not calculate the future value and use that to interpolate backward? This is the implicit timestepping algorithm, also known as backward Euler time integration, and it does not have the instability associated with the explicit algorithm.

To revisit the example of finite difference simulation of heat conduction, the explicit discretization looks like:

$$\frac{T_{i,n+1} - T_{i,n}}{\Delta t} = \nu \frac{T_{i-1,n} - 2T_{i,n} + T_{i+1,n}}{\Delta x^2} + f_i, \quad (2.10)$$

where f_i is the normalized source term $s_i/\rho c$ at grid point i . With implicit finite differencing, the time indices on the right side change:

$$\frac{T_{i,n+1} - T_{i,n}}{\Delta t} = \nu \frac{T_{i-1,n+1} - 2T_{i,n+1} + T_{i+1,n+1}}{\Delta x^2} + f_i. \quad (2.11)$$

Of course, the new temperatures in timestep $n + 1$ are unknown, so how do we calculate these derivatives? The answer is that we don't, explicitly, but we use these as a set of simultaneous equations which we can solve using linear algebra. For the equation above, we can multiply by Δt and rearrange to give:

$$-\frac{\nu \Delta t}{\Delta x^2} T_{i-1,n+1} + \left(1 + 2\frac{\nu \Delta t}{\Delta x^2}\right) T_{i,n+1} - \frac{\nu \Delta t}{\Delta x^2} T_{i+1,n+1} = f_i \Delta t + T_{i,n}. \quad (2.12)$$

This is the equation for one specific interior node; using the mesh Fourier number definition $\text{Fo}_M = \nu \Delta t / \Delta x^2$, and setting temperature boundary conditions at x_0 and x_4 to $T_{0,BC}$ and $T_{4,BC}$ respectively, we can write this equation for an aggregate of nodes:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -\text{Fo}_M & 1 + 2\text{Fo}_M & -\text{Fo}_M & 0 & 0 \\ 0 & -\text{Fo}_M & 1 + 2\text{Fo}_M & -\text{Fo}_M & 0 \\ 0 & 0 & -\text{Fo}_M & 1 + 2\text{Fo}_M & -\text{Fo}_M \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} T_{0,n+1} \\ T_{1,n+1} \\ T_{2,n+1} \\ T_{3,n+1} \\ T_{4,n+1} \end{pmatrix} = \begin{pmatrix} T_{0,BC} \\ f_1 \Delta t + T_{1,n} \\ f_2 \Delta t + T_{2,n} \\ f_3 \Delta t + T_{3,n} \\ T_{4,BC} \end{pmatrix}. \quad (2.13)$$

This reduces our difference equations to a simple linear system which we can solve using linear algebra techniques, such as multiplying both sides by the inverse of the matrix on the left.

Implicit timestepping is unconditionally stable. If we take the limit as Δt goes to infinity, we can divide equations 2.12 and 2.13 by the mesh Fourier number, so the 1s in the matrix diagonal and the $T_{i,n}$ on the right side vanish, and we obtain the steady-state result. The downside is that it is considerably more complex to implement than explicit timestepping.

Furthermore, this approach leads to errors on the order of the timestep size. But it is not hard to obtain quadratic or higher accuracy using both explicit and implicit time stepping algorithms known as Adams-Bashforth and Adams-Moulton methods, of which the next section presents one example.

3.2.3 Semi-Implicit Time Integration

Common to explicit and implicit timestepping is the assumption of constant time derivative throughout the timestep. If we acknowledge that the value of $\partial T/\partial t$ may be changing within the timestep, we can achieve better accuracy.

The simplest such approach, known as semi-implicit or Crank-Nicholson timestepping, involves linear interpolation of $\partial T/\partial t$ between timesteps n and $n+1$. Another way to look at this is that it estimates the value of $\partial T/\partial t$ at timestep $n + \frac{1}{2}$. Either way, for the heat equation, we average the right hand sides of equations 2.10 and 2.11 to give:

$$\frac{T_{i,n+1} - T_{i,n}}{\Delta t} = \frac{\nu T_{i-1,n} - 2T_{i,n} + T_{i+1,n} + T_{i-1,n+1} - 2T_{i,n+1} + T_{i+1,n+1}}{\Delta x^2} + \frac{f_{i,n} + f_{i,n+1}}{2}. \quad (2.14)$$

By treating the derivative as linear over the timestep, the error becomes second-order in the timestep size (that is, proportional to Δt^2). This is considerably more accurate than explicit or implicit timestepping, while more stable than explicit timestepping. However, because this average includes the previous timestep, this method is not unconditionally stable (as implicit timestepping is).

3.2.4 Adams-Bashforth

If first order accuracy of explicit and implicit methods are good (like straight Riemann integration), and second order semi-implicit is better (like trapezoid rule integration), why not go on to Simpson's rule for third or higher order discretizations? There are three approaches to accomplishing high-order accuracy in timestepping: an explicit method called Adams-Bashforth, an implicit method called Adams-Moulton, and a very efficient method with multiple function evaluations per timestep called Runge-Kutta.

This explicit higher-order method uses previous timesteps to extrapolate the $\partial T/\partial t$ curve into the next timestep. The simplest such method is just explicit timestepping, which is first-order in time, and can be expressed as:

$$u_{n+1} = u_n + \Delta t f(u_n), \quad (2.15)$$

where u is the unknown variable or set of variables (such as the values of temperature T_i), and

$$\frac{\partial u}{\partial t} = f(u). \quad (2.16)$$

This is analogous to equation 2.7. For higher-order accuracy, one can use previous timesteps, *e.g.* we can use this and the previous timestep to extrapolate a linear function of $\partial T/\partial t$ to achieve quadratic accuracy like semi-implicit timestepping; with uniform Δt , this is written generally as:

$$u_{n+1} = u_n + \Delta t \left[\frac{3}{2} f(u_n) - \frac{1}{2} f(u_{n-1}) \right]. \quad (2.17)$$

In general, for polynomial order p :

$$u_{n+1} = u_n + \Delta t \sum_{j=0}^{p-1} a_{p,j} f(u_{n-j}), \quad (2.18)$$

where $\sum a_{p,j} = 1$ for any p . This has the advantages of being explicit and solvable without simultaneous equations, while also very accurate, and the function $f(u)$ need only be evaluated once per timestep. The disadvantage is that like explicit/forward Euler time stepping, this method is only stable for suitably small timesteps.

3.2.5 Adams-Moulton

This implicit cousin of Adams-Bashforth time integration interpolates from previous timesteps and the next to provide a polynomial fit estimating $\partial T/\partial t$. Again, the first-order accurate version is the same as implicit/backward Euler time stepping, and the second-order accurate version is identical to semi-implicit/Crank-Nicholson. For a third-order accurate version, we interpolate two old timesteps and the new one to give a quadratic polynomial estimate of $\partial T/\partial t$, which integrates to give third-order accuracy. With uniform Δt , this is written:

$$u_{n+1} = u_n + \Delta t \left[\frac{5}{12}f(u_{n+1}) + \frac{8}{12}f(u_n) - \frac{1}{12}f(u_{n-1}) \right]. \quad (2.19)$$

In general for polynomial order p :

$$u_{n+1} = u_n + \Delta t \sum_{j=0}^{p-1} a_{p,j} f(u_{n-j+1}) \quad (2.20)$$

This is more accurate and more stable than Adams-Bashforth, but as with the comparison between implicit and explicit timestepping, is harder to implement. There is one (non-)linear solution per timestep, with as many function evaluations as necessary to solve the system.

3.2.6 Runge-Kutta Integration

Runge-Kutta integration achieves high-order accuracy by evaluating the function at multiple points within each timestep. Each function evaluation requires no simultaneous equation solution, giving it this advantage over implicit and Adams-Moulton methods. Though a general method, the most often used version is fourth-order accurate, which looks like::

$$u_{n+1} = u_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4), \quad (2.21)$$

where:

$$k_1 = f(u_n), \quad (2.22)$$

$$k_2 = f\left(u_n + \frac{\Delta t}{2}k_1\right), \quad (2.23)$$

$$k_3 = f\left(u_n + \frac{\Delta t}{2}k_2\right), \quad (2.24)$$

$$k_4 = f(u_n + \Delta t k_3). \quad (2.25)$$

This takes more time than explicit or Adams-Bashforth methods, particularly if function evaluations are expensive (which they are not for the heat equation), but doesn't require solution of any simultaneous equations.

3.3 Enthalpy Method

When a phase boundary is present in a system, it is necessary to change the formulation somewhat to account for it. Fortunately, for liquid-solid systems there is a relatively simple method for doing

this called the *enthalpy method*, which is based on tracking the enthalpy change at the liquid-solid interface.

The enthalpy uses the relation $\Delta H = \rho c_p \Delta T$, where c_p is the constant-pressure heat capacity. The 1-D enthalpy conservation equation goes:

$$\frac{\partial H}{\partial t} = k \frac{\partial^2 T}{\partial x^2}. \quad (3.26)$$

This is very similar to equation 2.1.

In the enthalpy method, we turn that differential equation directly into a difference equation; with explicit timestepping, this looks like:

$$\begin{aligned} \frac{H_{i,n+1} - H_{i,n}}{\Delta t} &= \frac{[k \frac{\partial T}{\partial x}]_{i+\frac{1}{2},n} - [k \frac{\partial T}{\partial x}]_{i-\frac{1}{2},n}}{\Delta x} + s \\ &= \frac{k_{i+\frac{1}{2},n}(T_{i+1,n} - T_{i,n}) - k_{i-\frac{1}{2},n}(T_{i,n} - T_{i-1,n})}{\Delta x^2} + s. \end{aligned} \quad (3.27)$$

Keeping the $k_{i+\frac{1}{2},n}$ and $k_{i-\frac{1}{2},n}$ distinct is necessary because the two sides may be in the different phases with potentially quite different conductivities.

A basic implementation of this stores two fields: H and T . In each timestep, the new H values are calculated from the old H and the old neighboring T values. The new T values are then calculated from the new H values using the inverse of the $H(T)$ function.¹

This “basic implementation” in one dimension is straightforward to insert into a spreadsheet, as has been done for you on the MIT server. It is interesting to watch this simulation in action: during freezing, the moving liquid-solid interface appears to be “pinned” at a spatial location as the enthalpy decreases with no corresponding change in the temperature.

This explicit timestepping formulation (in equation 3.27) will have a stability criterion for the same reason as the non-enthalpy method, and in fact, if the two phases have the same properties, the stability criterion will be the same as equation 2.9. If the two phases have different properties, then the one with the larger ν will have the smaller critical Δt , and choosing that smaller Δt will satisfy the stability criterion in both phases. (Note that at the interface itself, the heat capacity is essentially infinite, so $\nu = 0$ and it is stable for any timestep size.)

3.4 Finite Volume Approach to Boundary Conditions

The finite volume approach is slightly different from finite differences, and I like it because of its elegant approach to presenting boundary conditions. Rather than thinking of the spatial discretization as a set of points, the finite volume approach considers a set of regions, or elements, and considers the average temperature or enthalpy in each element. The heat conservation equation in the enthalpy method can then be written as:

$$V \frac{\partial H}{\partial T} = - \sum A_i q_i + V s, \quad (4.28)$$

¹One may also avoid storing the T values by calculating those of the previous timestep from H on the fly (though this makes post-processing the resulting data less straightforward). Although one can store only H and calculate T from that, in many cases one may *not* store only T and calculate H because H is not a unique function of T , *e.g.* in a pure material at the melting temperature, H can not be determined.

where V is the volume of the element (length in one dimension), the A_i are the areas of its faces (one in one dimension), and the q_i are the outward normal fluxes through those faces. At an interior point or volume, this and the finite difference method both give the same results, expressed in equations 2.5 and 3.27.

Based on this, one can very easily express a heat flux boundary condition, or a mixed condition where flux is a function of temperature. A common boundary condition of that type involves a *heat transfer coefficient* labeled as h :

$$\vec{q} \cdot \hat{n} = h(T - T_{env}), \quad (4.29)$$

where T_{env} is an environment temperature. This is often known as a convective boundary condition, based on boundary layer theory of transport in a fluid adjacent to a solid. These types of boundary conditions are straightforward to implement in finite volumes. Note however that this leads to a new stability criterion at the interface, which will put a limit on Δt similar to that in equation 2.9; derivation of this expression is left as an exercise to the reader.

What is not straightforward with this approach is setting a surface temperature. To do this requires setting the flux such that the temperature at the outside face is equal to the desired temperature. One can think of this in terms of a “virtual volume” outside the domain where the temperature is reflected through the desired surface temperature.

Referring again to the spreadsheet, its two boundary conditions are zero flux and convective, making this finite volume approach the logical one. You can see the finite volume implementation in the enthalpy elements on the left, and its boundary conditions on each side, but note that the widths of the boundary elements are half of those of the interior elements. The `castabox` software, also uses flux boundary conditions, based on both convective and radiative fluxes (both on the top surface, convective everywhere else). (`castabox` also has an interesting calculation for the stability criterion in three dimensions.)

Finally, note that the finite volume approach can be considered a “zero-order finite element” approach, whose first-order cousin will be introduced in the next chapter, with higher-order variants coming later.