Why the Code Solves the PDE

A Demonstration of Discretising Laplace's Equation

Paul Slavin
The University of Manchester
slavinp@cs.man.ac.uk

Overview

This document gives an informal explanation of *why* a computer implementation of a simple numerical discretisation 'solves' a particular PDE. It aims to familiarise students with the *concepts* of the Finite-Difference approach to numerical modeling and, by exploring the discretisation of a simple equation, to build *intuition* which may then be applied to more complex scenarios.

Introduction

Consider a hypothetical code[1] which implements a Finite-Difference solution of a two-dimensional Laplace's equation.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Several things will immediately be apparent about this code:

- No part of the code corresponds to the derivative operator, let alone the second partial.
- There is no global expression for the solution. Instead, the value at each point is related to that of the neighbouring values.
- The equation and its analytic solution are continuous in both space and time, yet the code necessarily uses discrete values for both.

Why then should this code be considered a valid representation of the original PDE? In what sense is the 'solution' that it calculates an accurate representation of the analytic solution?

[1] This document was originally prepared for students of the COMP6032 *"High Performance Computing in Science and Engineering"* course at the University of Manchester. A code equivalent to the 'hypothetical' code described herein is used as one of the lab exercises.

Discretisation

To answer these questions, we will consider a simplified version of the problem, illustrated in (1), which describes 1D linear advection. We will follow a sequence of steps which leads directly from this equation to the code which implements a finite-difference solution of it.

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{1}$$

We begin by considering the definition of the derivative.

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

This expresses the derivative of a function with respect to one of its variables as the instantaneous difference in its value induced by an infinitesimal change in the value of the variable in question. To consider how this statement may be translated into terms which can be represented in a computer, we *discretise* the equation (1) with respect to each of its variables. The function u in (1) is a function of both x, a single spatial dimension, and t representing time. As such, we introduce a coordinate system in which the possible values of x and t are represented by the indices i and n respectively, just as the possible values in a Cartesian coordinate system are represented by the indices x and y.

We wish to represent the *continuous* values of x and t in this coordinate system in a *discrete* form, so we define the concept of the 'smallest possible increment' Δ for each of these values. In spatial terms, the two values of x which can be considered to be adjacent are separated by one unit of Δx. Using our index notation, this is written as $_{i+\Delta x}$, or equivalently $_{i+1}$, and the change in the value of u induced by this change is expressed as $u_{i+1} - u_i$. Similarly we can denote values of t which are immediately 'adjacent' using the indices ^{n} and $^{n+1}$, so that the value of u for a particular x and t is denoted by u_i^n.

Using the definition of the derivative and applying our discrete notation to equation (1) gives...

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0$$

...which we can rearrange as...

$$u_i^{n+1} = u_i^n - c\frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n)$$

This provides an explicit expression for u_i^{n+1} in terms of values from the previous timestep. That is, by knowing the state of the system at time $_n$ we are able to calculate the state at $_{n+1}$, and so on.
This insight constitutes the essential technique of the finite-difference method: by expressing a PDE in a discretised form, we are able to calculate a solution by proceeding iteratively from a specified initial state.

Laplace's Equation

We now return to the example of Laplace's equation. In contrast to our simple advection example, Laplace's Equation involves second order derivatives, and we must therefore express these in a discretised form.
We use the symmetric form of the second derivative.

$$\frac{d^2f}{dx^2} = \lim_{h\to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

By applying our discretisation procedure to this and substituting the corresponding expressions into Laplace's equation, we arrive at a discretised representation of Laplace's equation.

$$\frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{\Delta x^2} + \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{\Delta y^2} = 0 \tag{2}$$

Whereas the advection equation we considered above contains a rate of change with respect to time, Laplace's equation represents a *steady state*. As such, the value of n in our discretisation can be considered to be a constant. While n may therefore be omitted entirely, we will see later that its inclusion gives us an insight into the manner of implementing an iterative solution.
We may then solve this discretised form for $u^n_{i,j}$.

$$\frac{2u^n_{i,j}}{\Delta x^2} + \frac{2u^n_{i,j}}{\Delta y^2} = \frac{u^n_{i+1,j} + u^n_{i-1,j}}{\Delta x^2} + \frac{u^n_{i,j+1} + u^n_{i,j-1}}{\Delta y^2} \tag{3a}$$

$$\frac{2u^n_{i,j}(\Delta x^2 + \Delta y^2)}{\Delta x^2 \Delta y^2} = \frac{(u^n_{i+1,j} + u^n_{i-1,j})\Delta y^2 + (u^n_{i,j+1} + u^n_{i,j-1})\Delta x^2}{\Delta x^2 \Delta y^2} \tag{3b}$$

$$u^n_{i,j} = \frac{(u^n_{i+1,j} + u^n_{i-1,j})\Delta y^2 + (u^n_{i,j+1} + u^n_{i,j-1})\Delta x^2}{2(\Delta x^2 + \Delta y^2)} \tag{3c}$$

Equation (3c) gives an explicit form for $u^n_{i,j}$ in terms of its neighbouring values.
Where $\Delta x = \Delta y$ the discretised Laplace equation in (2) reduces to

$$u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j} + u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1} = 0$$

Which may be solved directly for $u^n_{i,j}$ to give

$$u^n_{i,j} = \frac{(u^n_{i-1,j} + u^n_{i+1,j} + u^n_{i,j-1} + u^n_{i,j+1})}{4}$$

It will be observed that this explicit discretised equation corresponds exactly to an implementation in code of the finite-difference method for Laplace's equation.

Addendum: Further Investigation

This section introduces some additional concepts and vocabulary which may be investigated further to develop a deeper understanding of the material presented above.

Convergence and Stability

We have provided a plausible rationale for accepting that a discretised Finite-Difference scheme does indeed solve the PDE that it represents. But *must* such a technique always give the correct answer? The answer to this is the negative. Not only may a Finite-Difference scheme give an answer so inaccurate as to be useless, but it is not guaranteed to give any answer at all.

The conditions that must be satisfied for a scheme to *converge* upon an answer are described by the *Courant-Friedrichs-Lewy (CFL) condition.* This relates the size of an increment in the discretisation's spatial dimensions to the size of an increment in its timestep and to the speed of movement that takes place within a model. The CFL condition must be satisfied in order for a model to converge, but it should be noted that meeting the CFL condition is not sufficient to guarantee convergence for all models.

The accuracy of a Finite-Difference scheme may be measured by the techniques of *Von Neumann stability analysis.* The numerical stability of a scheme relates to the tendency of its *error*, defined as the differences between the computed solution and the analytical solution, to remain bounded as its timesteps advance. Von Neumann stability analysis decomposes this error into Fourier series, the relation between which provides a necessary and sufficient condition for a model's error to remain bounded.

Scheme Naming-conventions

The Finite-Difference scheme developed in this document is described as an *explicit* scheme as it provides an explicit formula for $u_{i,j}^n$ in terms of other values. An *implicit* scheme arrives at the desired value by establishing an equation which incorporates this value as a solution. By taking the average of both and explicit and implicit scheme, a *semi-implicit* or *Crank-Nicholson* scheme is produced.
Schemes may also be categorised as *Forward, Backwards, or Central* difference schemes. This simply relates to the form of the differential that is being represented by the discretisation; that is a right, left, or symmetric differential. Similarly, the terms *Upwind* and *Downwind* describe the direction in which amplitudes propagate in a scheme.