Mech 510
Computational Methods in Transport Phenomena I

Carl Ollivier-Gooch

UBC Department of Mechanical Engineering

Contents

List of Figures

Chapter 1

Intro to CFD

Strengths of Computational/Experimental/Analytic Fluid Dynamics

1.1 Modeling

Modeling is the process of separating important from unimportant physical effects in the physics of the problem to arrive at a mathematical model that is not too complex.

As a post-doc, I worked with a company that made its living by selling NO_x reduction systems for commercial tire incinerators. Ground-up tires were dumped in and burned. Gases passed by several heat exchangers to boil and superheat water for power generation. Because of the combustion conditions, there was a significant amount of nitrogen oxides (NO_x) in the flue gases, which was environmentally unacceptable. They needed an accurate CFD model that could be easily applied to a variety of incinerators rather quickly so that they could design NO_x reduction systems. In particular, they needed to predict temperature, velocity, and NO_x concentration both with and without their emission reduction system for several operating conditions. They needed to do this quickly (a couple of weeks at most).

Key features of the problem physics that would be considered in a full model of the problem:

- Turbulence and turbulent mixing

- Chemically reacting
- Heat transfer, both convective and radiative
- Very low Mach numbers
- Sprays (evaporating droplets)
- Complex geometry
- Flow separation (possibly unsteady)

Governing equations that include most of these effects:

Global continuity:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = S_\rho$$

Momentum:

$$\frac{\partial (\rho \vec{u})}{\partial t} + \nabla \cdot \left(\rho \vec{u} \otimes \vec{u} + P\vec{I} - \vec{\tau}_{ij} \right) = S_{\text{mom}}$$

Energy:

$$\frac{\partial E}{\partial t} + \nabla \cdot (\vec{u}(E+P)) = \frac{\partial Q}{\partial t} + \nabla \cdot (k \nabla T) + q_{\text{rad}} + \nabla \cdot (\vec{\tau}_{ij} \cdot \vec{u})$$

Species continuity:

$$\frac{\partial \rho_i}{\partial t} + \nabla \cdot (\rho_i \vec{u}) = S_{\rho_i}$$

Turbulence closure model

Droplet transport and evaporation model.

Total: Perhaps twenty PDE's with a wide range of time scales (from very fast chemical reactions to viscous diffusion and convection scales).

Including all of this physical detail in a computer model would make for a tremendously complicated and probably tremendously slow program. The essence of modeling is to balance physical fidelity against human and computer resources available. Generally, we use either the simplest model that gives a reasonable answer or the most complex model that can be programmed and run with available resources.

Assume for this problem that:

- Combustion can be modeled as a distributed heat source
- Sprays have a negligible mass, momentum, and energy effect on the flow
- Chemistry of NO_x reduction can be de-coupled (solved separately, *a posteriori*, based on computed temperature and velocity fields)
- Neglect radiative heat transfer (because radiative heat transfer is a non-local process)

This reduces the mathematical description of the problem to:

Global continuity:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0$$

Momentum:

$$\frac{\partial (\rho \vec{u})}{\partial t} + \nabla \cdot \left(\rho \vec{u} \otimes \vec{u} + P\vec{I} - \vec{\tau}_{ij} \right) = 0$$

Energy:

$$\frac{\partial E}{\partial t} + \nabla \cdot (\vec{u}(E+P)) = \nabla \cdot (k \nabla T) + \nabla \cdot (\vec{\tau}_{ij} \cdot \vec{u}) + \frac{\partial Q}{\partial t}$$

Species continuity:

Removed from main model

Turbulence closure model

Droplet transport and evaporation model.

Chemical reactions in the NO_x reduction process.

Total: Seven PDE's (with 2-eq turbulence model) plus a de-coupled set of PDE's to be solved separately for the droplets, etc, once velocities and temperatures are known.

1.2 Discretization

Modeling gives a system of PDE's to be solved. Only very rarely can we obtain an exact solution to these PDE's. Before we can compute a solution, we first must decide *where* we want to solve the equations. This requires us to generate a *mesh* containing a finite number of locations where we will solve the PDE's. Mesh generation is not a topic that we will discuss in Mech 510.

Once we have a mesh, we need to develop a representation of the PDE's on this mesh, including a time-evolution scheme. There are three main families of techniques for this:

Finite difference. Solution is represented by point values at mesh points. Replace each differential term in the PDE by a corresponding finite difference approximation. (See Figure 1.1)

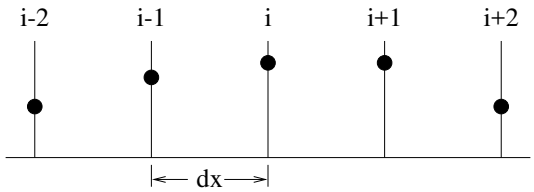


Figure 1.1: Schematic representation of finite difference approximation to a continuous solution.

- The original approach for CFD.
- Easy to get high-order discretizations (use high-order finite difference approximations).
- Doesn't necessarily conserve mass, momentum, and energy exactly.
- Impractical for unstructured meshes.

Finite volume. Solution is represented by control volume averages. Write the PDE's in volume integral form. Discretization based on evaluation of volume integral over small control volumes. (See Figure 1.2)

- Conserves mass, momentum, and energy exactly.

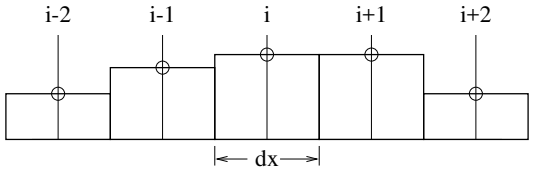


Figure 1.2: Schematic representation of finite volume approximation to a continuous solution.

- Applicable to any mesh topology w/ appropriate control volumes.
- Not too hard to get high-order discretizations (locally construct high-order polynomial representation of solution).

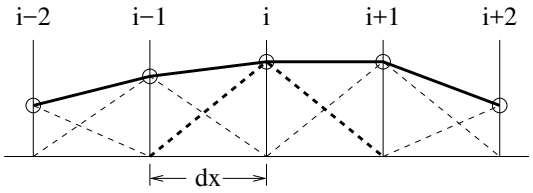


Figure 1.3: Schematic representation of finite element approximation to a continuous solution.

Finite element. Solution is represented by local *basis functions*. Multiply by a *test function* and integrate. Discretization based on evaluating integrals with given test and basis functions. (See Figure 1.3)

- Applicable to any mesh topology w/ appropriate test and basis functions.
- Lots of theoretical results showing convergence and stability of method.
- Not too hard to get high-order discretizations (high-order basis and test functions).
- Conservation of mass, momentum, and energy isn't automatic, but not too hard, either.

Discontinuous Galerkin FE methods have independent FE representations of the solution in each element, and so require both the FE device of multiplication by a test function (to account for variations within elements) and the FV device of calculating fluxes between elements.

In both Mech 510 and 511, we will focus on finite-volume methods. These have a more intuitive basis in flow physics than finite-difference methods, extend more easily to curvilinear and unstructured meshes, and correctly conserve quantities that are physically conserved. Finite-element methods have a number of mathematical advantages, but they are also much more mathematical and much less intuitive.

1.3 Accuracy and Stability

The finite representation of the PDE is not exact; some error is introduced regardless of how precisely we solve the discretized equations. Before we bother coding up a scheme, we want to analyze its accuracy so that we'll know what we're getting. The analysis gives an idea about how much difference there will be between our discrete solution (on a computer with infinite precision) and the exact solution to the PDE. Technically, the error estimates we will produce will be estimates of the *truncation error*: the amount by which our discrete approximation to the PDE fails to match the PDE. We also find out how the truncation error will change as we add mesh points. Our measurements of error will primarily be of the *discretization error*: the amount by which our discrete solution fails to match the exact solution of the PDE. Not surprisingly, truncation error and discretization error are related.

Also, for unsteady problems, we analyze the time-evolution scheme to determine its stability. That is, we determine whether errors in the solution will grow exponentially in time or remain bounded.

1.4 Verification and Validation

At this point, we have a complete description of the discrete scheme we will use to approximately solve our PDE. So we write a program to solve the discrete problem. We compile it. It doesn't compile the first few times. Finally it does. We run it. It crashes. Finally it runs and gives an answer. Should we believe this answer?

No. Absolutely not.

The output could be literally anything, from Egyptian hieroglyphics to the Martian alphabet; these are about as likely as getting the right solution the first try, in my experience. No CFD program should be considered correct until it has been thoroughly tested and debugged.

There are two interrelated parts to fixing a broken CFD program. *Verification* tells us whether the solutions we get for a series of simple test cases are correct, in the sense of getting the correct solution for the modeled equations we intended to solve. *Debugging* is the process of identifying *why* a program failed a test case and fixing it.

Once the code has been verified, we know that we're solving the equations properly. What we don't know without experimental data is whether our model is sufficient for the physical problem we're trying to solve. That level of testing is called *validation*: confirming that we're solving the right equations.

A testing plan should begin with ridiculously simple test cases and work up to test cases that are as near as possible in complexity to the problem to be solved. In this process, the early tests will be verification tests, and the later ones will be validation tests.

- Begin by testing code at the component level. While it's possible to debug 1000 lines of code (about the limit of program size for this course, typically) in one big piece, it's much easier to work with much smaller chunks. Basically, if you can define a task that a chunk of code is supposed to do, you can define a test that confirms that it was done correctly. Writing the test *first* is not necessarily a bad idea — then you'll know for sure when you're done. Ideally, you'll write unit tests for small subroutines to confirm (for instance) that your flux calculations are correct for specific input data for which you've calculated the exact flux. The more of these tests you write, the more confident you'll be that your code is correct.

- When testing the entire code by solving flow problems,

  - Each test case should have a known solution, whether analytic, experimental, or computed by a previously-verified program.
  - Each test case should ideally test a single new part of the physics or a single new interaction between already validated parts. This approach minimizes the number of places one must look for errors when the program gives an incorrect result for a test case; this more than offsets the time consumed in running more test cases. While it is nearly impossible to test only one thing with each case, the closer we come to devising such a plan, the easier it will be to validate and debug our program.

1.5 Efficiency

Now the program works, and we believe that the physics it simulates is adequate for our real world problem. Is the code efficient enough to be usable? Let's say that a run for this tire incinerator takes 20 CPU hours on the fastest machine available. For one run, that would be fine. But in the design context, we have to check a number of different operating conditions, which starts to get expensive. And for the company to stay in business, we have to design an emission reduction system for one of these things every week or so. So in this case, 20 hours isn't good enough. We have to go back and do one of several things:

- Simplify the physical model even more
- Make the discretization more efficient (this may mean more complicated)
- Improve the technique we use to solve the discretized equations
- Write more efficient code
- Buy a faster computer

Whatever we do, we have to sure that the final solution is still accurate enough.

1.6 Convergence

Finally, for any problem, we need to be sure that we have adequately resolved all of the important physical features of the flow. "Important" depends on the physical quantities we're after. If all we care about is NO_x mass fraction at the stack outflow, then we probably do not need to be concerned with resolving the length scales of turbulent eddies. To know this, we need either enough experience to know in advance how fine a mesh to use or to perform a *mesh refinement study*. In a mesh refinement study, we compute the solution on a series of progressively finer meshes until the physical quantity in which we are interested stops changing. This amounts to an empirical measurement of when discretization error is acceptably small.

Chapter 2

Modeling Based on the Navier-Stokes Equations

Most problems in computational fluid dynamics and computational heat transfer hinge on solving the Navier-Stokes equations, which describe viscous fluid flow, often in conjunction with auxiliary equations describing other physical phenomena, like turbulence, combustion, transport of chemical species, etc. Before considering such complicated cases, we will begin by examining the Navier-Stokes equations in detail, including non-dimensionalizing the basic equations and deriving some simple model problems based on that non-dimensional form.

2.1 Non-dimensionalization of the Navier-Stokes equations

Learning Objectives. Students will be able to:

- Non-dimensionalize the incompressible Navier-Stokes equations and the incompressible energy equation.
- Describe the conditions under which some terms drop out of the non-dimensional equations.

First, we write the Navier-Stokes equations (including the energy equation) in two dimensions for the case of constant coefficients:

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y} = 0 \tag{2.1}$$

$$\frac{\partial u}{\partial t}+\frac{\partial u^2}{\partial x}+\frac{\partial uv}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial x}+\nu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{2.2}$$

$$\frac{\partial v}{\partial t}+\frac{\partial uv}{\partial x}+\frac{\partial v^2}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial y}+\nu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right) \tag{2.3}$$

$$\begin{aligned}\frac{\partial T}{\partial t}+u\frac{\partial T}{\partial x}+v\frac{\partial T}{\partial y} &= \frac{k}{\rho c_p}\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right) \\ &\quad +\frac{\nu}{c_p}\left(2\left(\frac{\partial u}{\partial x}\right)^2+2\left(\frac{\partial v}{\partial y}\right)^2+\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right)^2\right)\end{aligned} \tag{2.4}$$

Note that the momentum equations have been written in conservation-law form by using the continuity equation. The same thing could have been done for the energy equation, but this equation is generally solved separately, with the velocity field already known; this makes it much less important to have this equation in conservation law form.

To non-dimensionalize Equations 2.1–2.4, we need reference values for length, velocity, pressure, and temperature (density is fixed, so we don't need a reference value for density). Suppose that we choose to non-dimensionalize length by L, velocity by u_{ref}, pressure by ρu_{ref}^2, and temperature by T_{ref}. Basically, we just assume that we can find some appropriate reference values L, u_{ref} and T_{ref} for whatever problem we're solving and that the pressure changes in the flow can be non-dimensionalized appropriately by the dynamic pressure associated with u_{ref}. If we do this, we can write the dimensional variables in terms of non-dimensional variables (with *) and reference values:

$$\begin{aligned} t &= t^*\frac{L}{u_{\text{ref}}} \\ x &= x^*L \\ y &= y^*L \\ u &= u^*u_{\text{ref}} \\ v &= v^*u_{\text{ref}} \\ P &= P^*\rho u_{\text{ref}}^2 \\ T &= T^*T_{\text{ref}} \end{aligned}$$

Substituting these into the continuity equation:

$$\frac{\partial\left(u^{*}u_{\text{ref}}\right)}{\partial\left(x^{*}L\right)}+\frac{\partial\left(v^{*}u_{\text{ref}}\right)}{\partial\left(x^{*}L\right)}=0$$

or

$$\frac{u_{\text{ref}}}{L}\left(\frac{\partial u^{*}}{\partial x^{*}}+\frac{\partial v^{*}}{\partial y^{*}}\right)=0$$

or

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

Substituting into the x-momentum equation:

$$\frac{\partial\left(u^{*}u_{\text{ref}}\right)}{\partial\left(t^{*}L/u_{\text{ref}}\right)}+\frac{\partial\left(u^{*^{2}}u_{\text{ref}}^{2}\right)}{\partial\left(x^{*}L\right)}+\frac{\partial\left(u^{*}v^{*}u_{\text{ref}}^{2}\right)}{\partial\left(y^{*}L\right)}=-\frac{1}{\rho}\frac{\partial\left(P^{*}\rho u_{\text{ref}}^{2}\right)}{\partial\left(x^{*}L\right)}+\nu\left(\frac{\partial^{2}\left(u^{*}u_{\text{ref}}\right)}{\partial\left(x^{*^{2}}L^{2}\right)}+\frac{\partial^{2}\left(u^{*}u_{\text{ref}}\right)}{\partial\left(y^{*^{2}}L^{2}\right)}\right)$$

Dividing all terms by u_{ref}^{2}/L,

$$\frac{\partial u^{*}}{\partial t^{*}}+\frac{\partial u^{*^{2}}}{\partial x^{*}}+\frac{\partial u^{*}v^{*}}{\partial y^{*}}=-\frac{\partial P^{*}}{\partial x^{*}}+\frac{\nu}{Lu_{\text{ref}}}\left(\frac{\partial^{2}u^{*}}{\partial x^{*^{2}}}+\frac{\partial^{2}u^{*}}{\partial y^{*^{2}}}\right)$$

where of course $\frac{\nu}{Lu_{\text{ref}}}\equiv\frac{1}{\text{Re}}$. Not surprisingly, a similar result holds for the y-momentum equation:

$$\frac{\partial v^{*}}{\partial t^{*}}+\frac{\partial u^{*}v^{*}}{\partial x^{*}}+\frac{\partial v^{*^{2}}}{\partial y^{*}}=-\frac{\partial P^{*}}{\partial y^{*}}+\frac{\nu}{Lu_{\text{ref}}}\left(\frac{\partial^{2}v^{*}}{\partial x^{*^{2}}}+\frac{\partial^{2}v^{*}}{\partial y^{*^{2}}}\right)$$

If we substitute the non-dimensional versions of the variables into the energy equation, we get:

$$\begin{aligned}\frac{\partial\left(T^{*}T_{\text{ref}}\right)}{\partial\left(t^{*}L/u_{\text{ref}}\right)}+u^{*}u_{\text{ref}}\frac{\partial\left(T^{*}T_{\text{ref}}\right)}{\partial\left(x^{*}L\right)}+v^{*}u_{\text{ref}}\frac{\partial\left(T^{*}T_{\text{ref}}\right)}{\partial\left(y^{*}L\right)} &= \\ \frac{k}{\rho c_{p}}\frac{T_{\text{ref}}}{L^{2}}\left(\frac{\partial^{2}T^{*}}{\partial x^{*^{2}}}+\frac{\partial^{2}T^{*}}{\partial y^{*^{2}}}\right) & \\ +\frac{\nu}{c_{p}}\frac{u_{\text{ref}}^{2}}{L^{2}}\left(2\left(\frac{\partial u^{*}}{\partial x^{*}}\right)^{2}+2\left(\frac{\partial v^{*}}{\partial y^{*}}\right)^{2}+\left(\frac{\partial v^{*}}{\partial x^{*}}+\frac{\partial u^{*}}{\partial y^{*}}\right)^{2}\right) &\end{aligned}$$

Dividing by $u_{\text{ref}}T_{\text{ref}}/L$, we get:

$$\begin{aligned}\frac{\partial T^*}{\partial t^*}+u^*\frac{\partial T^*}{\partial x^*}+v^*\frac{\partial T^*}{\partial y^*} &= \frac{k}{\rho c_p L u_{\text{ref}}}\left(\frac{\partial^2 T^*}{\partial x^{*2}}+\frac{\partial^2 T^*}{\partial y^{*2}}\right)\\ &+ \frac{\nu u_{\text{ref}}}{c_p T_{\text{ref}} L}\left(2\left(\frac{\partial u^*}{\partial x^*}\right)^2+2\left(\frac{\partial v^*}{\partial y^*}\right)^2\right.\\ &\qquad \left.+\left(\frac{\partial v^*}{\partial x^*}+\frac{\partial u^*}{\partial y^*}\right)^2\right)\end{aligned}$$

What are the dimensionless parameters here?

$$\frac{\rho L u_{\text{ref}} c_p}{k}=\frac{\rho L u_{\text{ref}}}{\mu}\frac{\mu c_p}{k}=\text{Re}\cdot\text{Pr}=\frac{\text{inertia}}{\text{viscosity}}\frac{\text{dissipation}}{\text{conduction}}$$

and

$$\frac{c_p T_{\text{ref}} L}{u_{\text{ref}}\nu}=\frac{L u_{\text{ref}}}{\nu}\frac{c_p T_{\text{ref}}}{u_{\text{ref}}^2}=\text{Re}\cdot\frac{1}{\text{Ec}}=\frac{\text{inertia}}{\text{viscosity}}\frac{\text{enthalpy}}{\text{kinetic energy}}$$

Summarizing the non-dimensional equations,

$$\frac{\partial u^*}{\partial x^*}+\frac{\partial v^*}{\partial y^*} = 0 \tag{2.5}$$

$$\frac{\partial u^*}{\partial t^*}+\frac{\partial u^{*2}}{\partial x^*}+\frac{\partial u^* v^*}{\partial y^*} = -\frac{\partial P^*}{\partial x^*}+\frac{1}{\text{Re}}\left(\frac{\partial^2 u^*}{\partial x^{*2}}+\frac{\partial^2 u^*}{\partial y^{*2}}\right) \tag{2.6}$$

$$\frac{\partial v^*}{\partial t^*}+\frac{\partial u^* v^*}{\partial x^*}+\frac{\partial v^{*2}}{\partial y^*} = -\frac{\partial P^*}{\partial y^*}+\frac{1}{\text{Re}}\left(\frac{\partial^2 v^*}{\partial x^{*2}}+\frac{\partial^2 v^*}{\partial y^{*2}}\right) \tag{2.7}$$

$$\begin{aligned}\frac{\partial T^*}{\partial t^*}+u^*\frac{\partial T^*}{\partial x^*}+v^*\frac{\partial T^*}{\partial y^*} &= \frac{1}{\text{Re}\cdot\text{Pr}}\left(\frac{\partial^2 T^*}{\partial x^{*2}}+\frac{\partial^2 T^*}{\partial y^{*2}}\right)\\ &+ \frac{\text{Ec}}{\text{Re}}\left(2\left(\frac{\partial u^*}{\partial x^*}\right)^2+2\left(\frac{\partial v^*}{\partial y^*}\right)^2+\left(\frac{\partial v^*}{\partial x^*}+\frac{\partial u^*}{\partial y^*}\right)^2\right)\end{aligned} \tag{2.8}$$

Note the extreme similarity in form between Equations 2.1–2.4 on the one hand and Equations 2.5–2.8 on the other. From now on, we'll use the non-dimensional form without the * superscripts.

Finally, it's worth noting that the non-dimensional parameters depend only on fluid properties (which we are assuming to be fixed) and on the reference values:

$$\begin{aligned} \mathrm{Re} &= \frac{\rho L u_{\mathrm{ref}}}{\mu} = \frac{L u_{\mathrm{ref}}}{\nu} \\ \mathrm{Pr} &= \frac{\mu c_p}{k} \\ \mathrm{Ec} &= \frac{u_{\mathrm{ref}}^2}{c_p T_{\mathrm{ref}}} \end{aligned}$$

We can deduce several things from the way in which these non-dimensional coefficients appear in the non-dimensional equations.

- The viscous terms in the momentum equations will be important unless the Reynolds number is extremely large, and these terms will dominate the momentum equations in the limit of low Reynolds number (creeping flow). The heat conduction and viscous dissipation terms in the energy equation also have Reynolds number scaling, with the same consequences.
- The heat conduction term has an additional dependence on the Prandtl number, which is a fluid property that measures whether momentum or heat diffuses more rapidly in the fluid. Prandtl number can range from about 10^{-2} in liquid metals to about 10^4 in oils and liquid polymers. A low value of Prandtl number indicates that thermal diffusivity dominates momentum diffusivity.
- The viscous dissipation has an additional dependence on the Eckert number, which is a measure of the relative importance of internal energy and kinetic energy in the flow.

2.2 Derivation of model problems

Although the Navier-Stokes equations are useful for solving physical problems, there are too many complexities involved in their solution for them to be a good starting point for study. However, we can derive several pedagogically useful model problems from the Navier-Stokes equations that can be used to illustrate particular techniques in CFD.

Learning Objectives. Students will be able to:

- Identify three model problems derived from the Navier-Stokes equations and classify them mathematically.

Poisson's Equation

Begin with the two-dimensional incompressible energy equation, including a source term:

$$\frac{\partial T}{\partial t}+u\frac{\partial T}{\partial x}+v\frac{\partial T}{\partial y}=\frac{1}{Re\cdot Pr}\nabla^2 T+\frac{Ec}{Re}\left(2\left(\frac{\partial u}{\partial x}\right)^2+2\left(\frac{\partial v}{\partial y}\right)^2+\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right)^2\right)+\dot{Q}$$

Assume steady-state and zero velocity:

$$\begin{aligned}\frac{1}{Re\cdot Pr}\nabla^2 T &= -\dot{Q}\\ \frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2} &= -Re\cdot Pr\,\dot{Q}\equiv S\end{aligned}$$

This is the familiar Poisson equation, which describes (among other things) steady heat conduction with a heat source. This is an elliptic PDE; that is, Poisson's equation poses a pure boundary value problem, with temperature everywhere coupled to temperature everywhere else.

Heat equation

Starting again with the incompressible energy equation in two dimensions, and this time assume zero velocity and no source term,

$$\begin{aligned}\frac{\partial T}{\partial t} &= \frac{1}{Re\cdot Pr}\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right)\\ &= \alpha\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right)\end{aligned}$$

This is the *transient heat conduction equation* or *heat equation*. This is a parabolic PDE, so the heat equation poses an initial-boundary value problem. The solution at (x,t) depends on the solution at all x at that time.

Wave equation

Begin yet again with the incompressible energy equation, and assume zero viscosity and thermal conductivity. Also, neglect the source term. Then we get:

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

If we know the velocity, then this is a hyperbolic PDE for the temperature T. This is the wave equation, which is an initial-value problem. For one dimension, we get

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} = 0$$

This is the linear convection equation in one dimension. This problem has a general solution of the form

$$E(x,ut) = f(x-ut)$$

so solutions travel unchanged at constant speed.

For what other bits of the physics of the Navier-Stokes equations is this a good model? That is, what other flow quantities are carried along with the flow?

Chapter 3

Space Discretization of PDE's

Learning Objectives. Students will be able to:

- Transform a partial differential equation from differential form into control volume form.
- Compute fluxes on a uniform mesh for any problem whose differential form contains no higher than second derivatives.
- Compute the flux integral for a control volume, given the fluxes at the boundaries of the CV.

Suppose we write our PDE with as many terms as possible in divergence form (the LHS of Eq. 3.1) with the remaining terms written as a source term (the RHS):

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} = S \tag{3.1}$$

This form is much more general than it looks. In particular, it is a simple matter to write the Navier-Stokes equations in this form, with no source term; turbulence and chemistry models (among other effects) add source terms.

Before we can compute the solution of this problem, we must rewrite the PDE into a system of algebraic equations relating the solution at one time level to the solution at the next time level. The first step in this process is space discretization, which will convert the PDE into a system of coupled ODE's describing the variation of solution unknowns with time. Next, these ODE's are discretized in time to produce a set of algebraic equations.

3.1 Overview

We begin with a comparison among finite difference, finite element, and finite volume methodologies. These methods can all be applied to the PDE in Eq. 3.1, but for simplicity and concreteness, we will consider the one-dimensional advection-diffusion equation:

$$\frac{\partial T}{\partial t} + \frac{\partial}{\partial x}\left(uT - \alpha\frac{\partial T}{\partial x}\right) = 0$$

where u and α are known constants. We consider the spatial domain $[0,1]$, divided into N equal intervals, with $T(0,t) = 1$ and $\frac{\partial T}{\partial x}(1,t) = 0$. Initial conditions need not concern us here.

3.1.1 The Finite Difference Method

In the finite difference methods, we compute the solution at points in the domain. In this case, we will have $N+1$ points located at $x_i = \frac{i}{N}$, $i = 0..N$. We will refer to the solution at x_i as T_i. To approximate the spatial derivatives, we will use finite differences, just as in the classical definition of the derivative. Because there is more than one way to approximate a derivative at a point, the discretization is not unique; one possibility is to use

$$\begin{aligned}\frac{\partial T}{\partial x} &\approx \frac{T_{i+1} - T_{i-1}}{2\Delta x} \\ \frac{\partial^2 T}{\partial x^2} &\approx \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2}\end{aligned}$$

Using Taylor series expansions, it is easy to verify that these approximations are accurate to within $O\left(\Delta x^2\right)$; that is, that the difference between $=$ and $\approx$ for these approximations decreases with the square of the mesh spacing. In this case, we can write a discrete approximation to the PDE as:

$$\frac{dT_i}{dt} + u\frac{T_{i+1} - T_{i-1}}{2\Delta x} = \alpha\frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2}$$

again to within $O\left(\Delta x^2\right)$. This leaves us $N-1$ equations (for points 1 through $N-1$) written in terms of $N+1$ unknowns (T_0 and T_N are the other two). Fortunately, we have two boundary conditions, which can again be written by replacing derivatives

with approximations:

$$\begin{aligned} T_0 &= 1 \\ \frac{T_N - T_{N-1}}{\Delta x} &= 0 \end{aligned}$$

The former is exact (no approximation was required), while the latter turns out to be first-order accurate.

3.1.2 The Finite Element Method

In the finite element method, the solution is computed at nodal values (here we use the same points as in the finite difference example), and interpolated between nodes by using basis functions so that a continuous representation of the solution is available. That is, the global solution is written as

$$T(x,t) = \sum_{i=0}^{N} b_i(x)\, T_i \tag{3.2}$$

where the b_i are *basis functions*, and the T_i are the nodal solution values (these vary in time, but this is suppressed for notational clarity). Basis functions are always defined to have a value of 1 at exactly one nodal point, and zero at all others; this ensures that the interpolation matches the nodal values at the nodes. Basis functions are also defined to have *compact support*, meaning that they are uniformly zero outside of a small region near "their" node. For our present purposes, we will consider the piecewise-linear tent-shaped basis function given by:

$$b_i(x) = \begin{cases} 1 + \frac{x - x_i}{\Delta x} & x_{i-1} \le x \le x_i \\ 1 - \frac{x - x_i}{\Delta x} & x_i \le x \le x_{i+1} \\ 0 & \text{elsewhere} \end{cases} \tag{3.3}$$

This basis function must be modified at the ends of the domain to be one sided, so that the basis function does not overlap the end of the domain.

Finite element discretization proceeds by multiplying the PDE by a test function $w_i(x)$; we will consider the Galerkin finite element discretization, in which the test and basis functions are identical. This weighted PDE is integrated over the domain, with the solution represented by Equation. 3.2. Repeating this for each basis function results in $N+1$ equations for the nodal solution values.

In this case, for an interior node, we write:

$$w_i \sum_{j=0}^{N} \left(b_j \frac{dT_j}{dt} \right) + u w_i \sum_{j=0}^{N} \left(T_j \frac{db_j}{dx} \right) = \alpha w_i \sum_{j=0}^{N} \left(T_j \frac{d^2 b_j}{dx^2} \right)$$

Note that, even for linear basis functions, the second derivative on the right-hand side is non-zero at $x = x_j$ (where it is infinite). Also, for the given basis and test functions, the only non-zero terms occur for $j = i-1, i, i+1$, which reduces both analytic and computational effort enormously. Now we integrate over the domain, which reduces to integration over the support of w_i:

$$\int_0^1 \left[w_i \sum_{j=i-1}^{i+1} \left(b_j \frac{dT_j}{dt} \right) + u w_i \sum_{j=i-1}^{i+1} \left(T_j \frac{db_j}{dx} \right) \right] dx = \int_0^1 \alpha w_i \sum_{j=i-1}^{i+1} \left(T_j \frac{d^2 b_j}{dx^2} \right) dx$$

$$\int_{(i-1)\Delta x}^{(i+1)\Delta x} \left[w_i \sum_{j=i-1}^{i+1} \left(b_j \frac{dT_j}{dt} \right) + u w_i \sum_{j=i-1}^{i+1} \left(T_j \frac{db_j}{dx} \right) \right] dx = \int_{(i-1)\Delta x}^{(i+1)\Delta x} \alpha w_i \sum_{j=i-1}^{i+1} \left(T_j \frac{d^2 b_j}{dx^2} \right) dx$$

Let's look at one term at a time. First, the advection term:

$$\int_{(i-1)\Delta x}^{(i+1)\Delta x} u w_i \sum_{j=i-1}^{i+1} \left(T_j \frac{db_j}{dx} \right) dx = \left[u w_i \sum_{j=i-1}^{i+1} \left(T_j b_j \right) \right]_{(i-1)\Delta x}^{(i+1)\Delta x} - \int_{(i-1)\Delta x}^{(i+1)\Delta x} u \frac{dw_i}{dx} \sum_{j=i-1}^{i+1} \left(T_j b_j \right) dx$$

Here we've used integration by parts, and note that the first term on the right is zero for all i (except for $i = N$, a boundary case which we'll ignore for now). In the second term, the derivative of the weight function is:

$$\frac{dw_i}{dx} = \begin{cases} \frac{1}{\Delta x} & (i-1)\Delta x < x < i\Delta x \\ \frac{-1}{\Delta x} & i\Delta x < x < (i+1)\Delta x \end{cases}$$

and the sum is the solution interpolant:

$$\sum_{j=i-1}^{i+1} T_j b_j = \begin{cases} T_{i-1} + \left(\frac{x}{\Delta x} - (i-1) \right) (T_i - T_{i-1}) & (i-1)\Delta x < x < i\Delta x \\ T_{i+1} + \left(i + 1 - \frac{x}{\Delta x} \right) (T_i - T_{i+1}) & i\Delta x < x < (i+1)\Delta x \end{cases}$$

So that integral gets split into two pieces, thus:

$$\begin{aligned} -\int_{(i-1)\Delta x}^{(i+1)\Delta x} u \frac{dw_i}{dx} \sum_{j=i-1}^{i+1} \left(T_j b_j \right) dx = & -\int_{(i-1)\Delta x}^{i\Delta x} u \frac{1}{\Delta x} \left(T_{i-1} + \left(\frac{x}{\Delta x} - (i-1) \right) (T_i - T_{i-1}) \right) dx \\ & -\int_{i\Delta x}^{(i+1)\Delta x} u \left(\frac{-1}{\Delta x} \right) \left(T_{i+1} + \left(i + 1 - \frac{x}{\Delta x} \right) (T_i - T_{i+1}) \right) dx \end{aligned}$$

$$\begin{aligned}
&= -\left[\frac{u}{\Delta x}\left(T_{i-1}x+\left(\frac{x^2}{2\Delta x}-(i-1)x\right)(T_i-T_{i-1})\right)\right]_{(i-1)\Delta x}^{i\Delta x} \\
&\quad -\left[\frac{-u}{\Delta x}\left(T_{i+1}x+\left((i+1)x-\frac{x^2}{2\Delta x}\right)(T_i-T_{i+1})\right)\right]_{i\Delta x}^{(i+1)\Delta x} \\
&= -\frac{u}{\Delta x}\left[T_{i-1}\Delta x+\left(\frac{(2i-1)\Delta x^2}{2\Delta x}-(i-1)\Delta x\right)(T_i-T_{i-1})\right] \\
&\quad +\frac{u}{\Delta x}\left[T_{i+1}\Delta x+\left((i+1)\Delta x-\frac{(2i+1)\Delta x^2}{2\Delta x}\right)(T_i-T_{i+1})\right] \\
&= u\left[-T_{i-1}-\frac{T_i-T_{i-1}}{2}+T_{i+1}+\frac{T_i-T_{i+1}}{2}\right] \\
&= u\frac{T_{i+1}-T_{i-1}}{2}
\end{aligned}$$

The second-last line contains the average values between $(i,i+1)$ and $(i,i-1)$ in a recognizable form; this isn't surprising, considering we integrated the solution times a constant.

For the diffusive term, we'll once again integrate by parts once:

$$\int_{(i-1)\Delta x}^{(i+1)\Delta x}\alpha w_i\sum_{j=i-1}^{i+1}\left(T_j\frac{d^2b_j}{dx^2}\right)dx=\left[\alpha w_i\sum_{j=i-1}^{i+1}\left(T_j\frac{db_j}{dx}\right)\right]_{(i-1)\Delta x}^{(i+1)\Delta x}-\alpha\int_{(i-1)\Delta x}^{(i+1)\Delta x}\frac{dw_i}{dx}\sum_{j=i-1}^{i+1}\left(T_j\frac{db_j}{dx}\right)dx$$

Again, except for boundary cases, the first term is zero for all i. The second term has a piecewise constant integrand, with

$$\sum_{j=i-1}^{i+1}T_j\frac{db_j}{dx}=\begin{cases}\frac{T_i-T_{i-1}}{\Delta x} & (i-1)\Delta x<x<i\Delta x\\ \frac{T_{i+1}-T_i}{\Delta x} & i\Delta x<x<(i+1)\Delta x\end{cases}$$

So that remaining integral becomes:

$$\begin{aligned}
-\alpha\int_{(i-1)\Delta x}^{(i+1)\Delta x}\frac{dw_i}{dx}\sum_{j=i-1}^{i+1}\left(T_j\frac{db_j}{dx}\right)dx &= -\alpha\left[\frac{1}{\Delta x}\frac{T_i-T_{i-1}}{\Delta x}+\left(\frac{-1}{\Delta x}\right)\frac{T_{i+1}-T_i}{\Delta x}\right]\Delta x \\
&= \alpha\frac{T_{i+1}-2T_i+T_{i-1}}{\Delta x}
\end{aligned}$$

Okay, that's two integrals out of three done. Now for the time-dependent term. Since the node locations and the basis and test functions are constant, we can pull

the time derivative out of the integral to get:

$$\int_{(i-1)\Delta x}^{(i+1)\Delta x} w_i \sum_{j=i-1}^{i+1} \left(b_j \frac{dT_j}{dt} \right) dx = \frac{d}{dt} \int_{(i-1)\Delta x}^{(i+1)\Delta x} w_i \sum_{j=i-1}^{i+1} \left(b_j T_j \right) dx$$

Now we substitute for w_i and b_j, and it's easy to get to:

$$\begin{aligned} \frac{d}{dt} \int_{(i-1)\Delta x}^{(i+1)\Delta x} w_i \sum_{j=i-1}^{i+1} \left(b_j T_j \right) dx &= \frac{d}{dt} \int_{(i-1)\Delta x}^{i\Delta x} \left(\frac{x}{\Delta x} - (i-1) \right) \left(T_{i-1} + (T_i - T_{i-1}) \left(\frac{x}{\Delta x} - (i-1) \right) \right) \\ &\quad + \frac{d}{dt} \int_{i\Delta x}^{(i+1)\Delta x} \left(i+1 - \frac{x}{\Delta x} \right) \left(T_{i+1} + (T_i - T_{i+1}) \left(i+1 - \frac{x}{\Delta x} \right) \right) \\ &= \Delta x \frac{d}{dt} \left(\frac{T_{i+1} + 4T_i + T_{i-1}}{6} \right) \end{aligned}$$

where the last line reflects not the occurrence of a miracle, but simply some tedious algebra. Combining the results of evaluating these various integrals, we get the linear Galerkin finite-element discretization for the advection diffusion equation:

$$\begin{aligned} \frac{\partial T}{\partial t} + \frac{\partial uT}{\partial x} &= \alpha \frac{\partial^2 T}{\partial x^2} \\ \left(\frac{1}{6} \frac{dT_{i+1}}{dt} + \frac{2}{3} \frac{dT_i}{dt} + \frac{1}{6} \frac{dT_{i-1}}{dt} \right) + u \frac{T_{i+1} - T_{i-1}}{2\Delta x} &= \alpha \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2} \end{aligned}$$

Note that we have a set of algebraic equations to solve for the time derivatives. For steady-state computations, it's customary to fold the left-hand side terms together to get simply dT_i/dt; this is a specific example of the general *lumped mass matrix* technique. For boundary conditions, we once again apply

$$\begin{aligned} T_0 &= 1 \\ \frac{T_N - T_{N-1}}{\Delta x} &= 0 \end{aligned}$$

although for unsteady problems, it's convenient to differentiate these with respect to time.

3.1.3 The Finite Volume Method

The finite volume method divides the domain into control volumes. In this case, there are N control volumes, with control volume i covering the region from $(i-1)\Delta x$

to $i\Delta x$. The quantity we compute in this case will be the control volume average of the solution, which we will refer to as $\bar{T}_i \equiv \frac{1}{\Delta x}\int_{(i-1)\Delta x}^{i\Delta x} T\,dx$. We begin by integrating the equations over a control volume:

$$\begin{aligned}\frac{\partial T}{\partial t}+\frac{\partial uT}{\partial x} &= \alpha\frac{\partial^2 T}{\partial x^2}\\ \int_{CV\,i}\frac{\partial T}{\partial t}dx+\int_{CV\,i}\frac{\partial uT}{\partial x} &= \int_{CV\,i}\alpha\frac{\partial^2 T}{\partial x^2}dx\end{aligned}$$

Now we apply Gauss's Theorem to convert the second and third integrals; in three dimensions, this is stated as:

$$\int_\Omega \nabla\cdot\vec{F}\,dV = \oint_{\partial\Omega}\vec{F}\cdot\hat{n}\,dA$$

where $\vec{F}$ is an arbitrary vector and $\hat{n}$ is an outward unit normal. Applying Gauss's theorem in this one-dimensional context, we get:

$$\begin{aligned}\int_{CV\,i}\frac{\partial T}{\partial t}dx+\int_{CV\,i}\frac{\partial uT}{\partial x}dx &= \int_{CV\,i}\alpha\frac{\partial^2 T}{\partial x^2}dx\\ \int_{CV\,i}\frac{\partial T}{\partial t}dx+(uT)_{x=i\Delta x}-(uT)_{x=(i-1)\Delta x} &= \alpha\left(\left(\frac{\partial T}{\partial x}\right)_{x=i\Delta x}-\left(\frac{\partial T}{\partial x}\right)_{x=(i-1)\Delta x}\right)\end{aligned}$$

For fixed control volumes, the derivative can be removed from the integral and converted to a complete differential. The remaining quantities in the discretization represent fluxes across the control volume boundaries, and sensible choices for computing these fluxes is a key to success with the finite volume method. In the case, we will write

$$\begin{aligned}(uT)_{x=i\Delta x} &= u\frac{\bar{T}_i+\bar{T}_{i+1}}{2}\\ \left(\frac{\partial T}{\partial x}\right)_{x=i\Delta x} &= \frac{\bar{T}_{i+1}-\bar{T}_i}{\Delta x}\end{aligned}$$

These choices, as we shall see, turn out to be second-order accurate. If we substitute these expressions into our control volume averaged PDE, we get

$$\begin{aligned}\int_{CV\,i}\frac{\partial T}{\partial t}dx+(uT)_{x=i\Delta x}-(uT)_{x=(i-1)\Delta x} &= \alpha\left(\left(\frac{\partial T}{\partial x}\right)_{x=i\Delta x}-\left(\frac{\partial T}{\partial x}\right)_{x=(i-1)\Delta x}\right)\\ \Delta x\frac{d\bar{T}_i}{dt}+u\frac{\bar{T}_{i+1}-\bar{T}_{i-1}}{2} &= \alpha\frac{\bar{T}_{i+1}-2\bar{T}_i+\bar{T}_{i-1}}{\Delta x}\\ \frac{d\bar{T}_i}{dt}+u\frac{\bar{T}_{i+1}-\bar{T}_{i-1}}{2\Delta x} &= \alpha\frac{\bar{T}_{i+1}-2\bar{T}_i+\bar{T}_{i-1}}{\Delta x^2}\end{aligned}$$

Again, we get an interior scheme indistinguishable from the finite difference and finite element schemes for this problem. The boundary conditions, however, differ. In this case, if we follow from the flux definitions, we find that the boundary conditions can be written as:

$$\begin{aligned} T_{x=0} &= \frac{\bar{T}_1 + \bar{T}_0}{2} = 1 \\ \left(\frac{\partial T}{\partial x}\right)_{x=N\Delta x} &= \frac{\bar{T}_{N+1} - \bar{T}_N}{\Delta x} = 0 \end{aligned}$$

This looks like it might be a step backwards (we've introduced two new variables, $\bar{T}_0$ and $\bar{T}_{N+1}$), except that the interior scheme contains these variables as well. For instance,

$$\frac{d\bar{T}_1}{dt} + u\frac{\bar{T}_2 - \bar{T}_0}{2\Delta x} = \alpha\frac{\bar{T}_2 - 2\bar{T}_1 + \bar{T}_0}{\Delta x^2}$$

So in the end, we can choose to think of this as a problem with $N+2$ equations and $N+2$ unknowns.

3.2 Transformation of a PDE into Control Volume Form

If we integrate Equation 3.1 over a three-dimensional control volume, we get

$$\begin{aligned} \int_{CV} \frac{\partial U}{\partial t} dV + \int_{CV} \frac{\partial F}{\partial x} dV + \int_{CV} \frac{\partial G}{\partial y} dV + \int_{CV} \frac{\partial H}{\partial z} dV &= \int_{CV} S\, dV \\ \int_{CV} \frac{\partial U}{\partial t} dV + \int_{CV} \left(\frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z}\right) dV &= \int_{CV} S\, dV \\ \int_{CV} \frac{\partial U}{\partial t} dV + \int_{CV} \nabla \cdot \vec{F}\, dV &= \int_{CV} S\, dV \end{aligned}$$

where the last equation arises by defining $\vec{F} = F\hat{i} + G\hat{j} + H\hat{k}$. Using Gauss's theorem, we get

$$\int_{CV} \frac{\partial U}{\partial t} dV + \oint_{\partial(CV)} \vec{F} \cdot \vec{n}\, dA = \int_{CV} S\, dV$$

If we assume that the size and shape of the control volume is fixed (computationally, assume that the mesh is not moving), we can simplify a bit further.

$$\frac{d}{dt}\int_{CV} U\, dV + \oint_{\partial(CV)} \vec{F} \cdot \vec{n}\, dA = \int_{CV} S\, dV \tag{3.4}$$

In the finite-volume method, we abandon hope of knowing anything about the details of the solution within a control volume and instead content ourselves with computing $\bar{U} \equiv \frac{1}{V}\int_{CV} U\,dV$. This average value is *not necessarily the value of the solution at any fixed point within the control volume*, including its centroid; forgetting this fact can lead to unfortunate misunderstandings when developing finite-volume algorithms.[1]

If we also define a mean source term contribution $\bar{S} \equiv \frac{1}{V}\int_{CV} S\,dV$, we can write Equation 3.4 as follows.

$$\frac{d\bar{U}}{dt} = -\frac{1}{V}\oint_{\partial(CV)} \vec{F}\cdot\vec{n}\,dA + \bar{S} \tag{3.5}$$

This equation states that the average value $\bar{U}$ of the solution in the control volume changes at a rate determined by the volume average of the net flux of stuff across the boundaries of the control volume $\frac{1}{V}\oint \vec{F}\cdot\vec{n}\,dA$ and the average rate of production of stuff inside the control volume $\bar{S}$.

Also, Equation 3.5 suggests that for a general time-varying problem, the process of advancing the solution from one time level $t = n\Delta t$ to the next ($(n+1)\Delta t$) requires four operations:

1. Evaluation of the flux $\vec{F}$ at the surface of the control volume.
2. Integration of the normal flux $\vec{F}\cdot\vec{n}$ around the boundary of the control volume.
3. Evaluation and integration of the source term S over the control volume.
4. Updating the control volume average value $\bar{U}$.

3.3 Second-order Accurate Flux for the Poisson Equation

Poisson's equation in two dimensions is:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = S.$$

[1] Nevertheless, it isn't hard to show (by expanding in a Taylor series and integrating over the control volume) that $\bar{U}$ is within $O(\Delta x^2)$ of U at the centroid of the control volume. Likewise, $\bar{S}$ can be evaluated to within $O\left(\Delta x^2\right)$ by taking its value as $\bar{S} \approx S(\bar{U})$.

Integrating over control volumes, we have

$$\int_{CV}\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right)dA = \int_{CV} S\,dA \tag{3.6}$$

$$\int_{CV}\nabla\cdot\begin{pmatrix}\frac{\partial T}{\partial x}\\ \frac{\partial T}{\partial y}\end{pmatrix}dA = \bar{S}A \tag{3.7}$$

$$\oint_{\partial CV}(\nabla T)\cdot\vec{n}\,ds = \bar{S}A \tag{3.8}$$

The last transformation uses Gauss's theorem. So the flux in Poisson's equation is $\begin{pmatrix}\frac{\partial T}{\partial x} & \frac{\partial T}{\partial y}\end{pmatrix}^T$. The normal component of this flux is $\frac{\partial T}{\partial x}$ on faces perpendicular to the x-axis and $\frac{\partial T}{\partial y}$ on faces perpendicular to the y-axis.

Recall that the derivative can be defined as

$$\left.\frac{dT}{dx}\right|_{x_0} = \lim_{\varepsilon\to 0}\frac{T(x_0+\varepsilon)-T(x_0-\varepsilon)}{2\varepsilon}, \tag{3.9}$$

assuming that the limit exists. This is the well-known centered difference formula. Note that the difference between the total and partial derivative here is simply that the partial derivative carries along a non-varying second independent variable:

$$\left.\frac{\partial T}{\partial x}\right|_{x_0} = \lim_{\varepsilon\to 0}\frac{T(x_0+\varepsilon,y)-T(x_0-\varepsilon,y)}{2\varepsilon}, \tag{3.10}$$

Our problem with applying this directly is that we don't have values of the solution like $T(x_0+\varepsilon)$ available to us; all we have are control volume averages. Can we use control volume averages in Eqs. 3.9 and 3.10 and still approximate the derivative accurately enough? That is, what is the error in approximating

$$\left.\frac{dT}{dx}\right|_{i+\frac{1}{2}} \approx \frac{\overline{T}_{i+1}-\overline{T}_i}{\Delta x}$$

To determine this, we need to use Taylor series and integration to express the control volume averages in control volumes i and $i+1$ in terms of the solution and its derivatives at the interface between these control volumes, where $x = x_{i+\frac{1}{2}}$. First,

for control volume $i+1$, we have:

$$\begin{aligned}
\overline{T}_{i+1} &= \frac{1}{\Delta x}\int_{x_{i+\frac{1}{2}}}^{x_{i+\frac{3}{2}}}\left(T_{i+\frac{1}{2}}+\left(x-x_{i-\frac{1}{2}}\right)\left.\frac{dT}{dx}\right|_{i+\frac{1}{2}}+\frac{\left(x-x_{i-\frac{1}{2}}\right)^2}{2}\left.\frac{d^2T}{dx^2}\right|_{i+\frac{1}{2}}+\frac{\left(x-x_{i-\frac{1}{2}}\right)^3}{6}\left.\frac{d^3T}{dx^3}\right|_{i+\frac{1}{2}}+\cdots\right)dx \\
&= \frac{1}{\Delta x}\int_0^{\Delta x}\left(T_{i+\frac{1}{2}}+\xi\left.\frac{dT}{dx}\right|_{i+\frac{1}{2}}+\frac{\xi^2}{2}\left.\frac{d^2T}{dx^2}\right|_{i+\frac{1}{2}}+\frac{\xi^3}{6}\left.\frac{d^3T}{dx^3}\right|_{i+\frac{1}{2}}+\cdots\right)d\xi \\
&= \frac{1}{\Delta x}\left(\xi T_{i+\frac{1}{2}}+\frac{\xi^2}{2}\left.\frac{dT}{dx}\right|_{i+\frac{1}{2}}+\frac{\xi^3}{6}\left.\frac{d^2T}{dx^2}\right|_{i+\frac{1}{2}}+\frac{\xi^4}{24}\left.\frac{d^3T}{dx^3}\right|_{i+\frac{1}{2}}+\cdots\right)_0^{\Delta x} \\
&= T_{i+\frac{1}{2}}+\frac{\Delta x}{2}\left.\frac{dT}{dx}\right|_{i+\frac{1}{2}}+\frac{\Delta x^2}{6}\left.\frac{d^2T}{dx^2}\right|_{i+\frac{1}{2}}+\frac{\Delta x^3}{24}\left.\frac{d^3T}{dx^3}\right|_{i+\frac{1}{2}}+\cdots
\end{aligned}$$

Note carefully that this is not the pointwise value of $T(x_i)$; all terms after the second are different for the control volume average than for the Taylor expansion of the function at that point. Should we need more terms for our accuracy analysis, we can always compute them. Similarly, we can write:

$$\overline{T}_i = T_{i+\frac{1}{2}}-\frac{\Delta x}{2}\left.\frac{dT}{dx}\right|_{i+\frac{1}{2}}+\frac{\Delta x^2}{6}\left.\frac{d^2T}{dx^2}\right|_{i+\frac{1}{2}}-\frac{\Delta x^3}{24}\left.\frac{d^3T}{dx^3}\right|_{i+\frac{1}{2}}+\cdots$$

Combining these,

$$\frac{\overline{T}_{i+1}-\overline{T}_i}{\Delta x} = \left.\frac{dT}{dx}\right|_{i+\frac{1}{2}}+\frac{\Delta x^2}{12}\left.\frac{d^3T}{dx^3}\right|_{i+\frac{1}{2}}+O\left(\varepsilon^4\right) \tag{3.11}$$

Another way to write this is to use *Taylor tables*. Basically, this approach is just a convenient way to avoid writing out all of every term each time you expand something in a Taylor series. Each column of the Taylor table represents one term in the Taylor series expansion, and each row represents an expression that is being expanded. The entries in the table are coefficients. Here's the previous example done using a Taylor table.

	$T(x_0)$	$\left.\frac{\partial T}{\partial x}\right\|_{x_0}$	$\left.\frac{\partial^2 T}{\partial x^2}\right\|_{x_0}$	$\left.\frac{\partial^3 T}{\partial x^3}\right\|_{x_0}$
$\frac{\overline{T}_{i+1}}{\Delta x}$	$\frac{1}{\Delta x}$	$\frac{1}{2}$	$\frac{\Delta x}{6}$	$\frac{\Delta x^2}{24}$
$-\frac{\overline{T}_i}{\Delta x}$	$-\frac{1}{\Delta x}$	$\frac{1}{2}$	$-\frac{\Delta x}{6}$	$\frac{\Delta x^2}{24}$
$\frac{\overline{T}_{i+1}-\overline{T}_i}{\Delta x}$	0	1	0	$\frac{\Delta x^2}{12}$

The *truncation error* in a difference approximation D of a differential operator $\mathbf{D}$ is defined to be $D-\mathbf{D}$.[2] An approximation is said to be k^{th}*-order accurate* if and only if the leading-order term in the truncation error is $O\left(\varepsilon^k\right)$.

For our example, the truncation error is $\frac{\Delta x^2}{12}\left.\frac{\partial^3 T}{\partial x^3}\right|_{x_0} + O\left(\varepsilon^4\right)$. This approximation is therefore second-order accurate, and the error in the approximation will fall by a factor of four each time ε is reduced by a factor of two.

Returning to our example of Poisson's equation, Equation 3.11 implies that we can write

$$\left.\frac{\partial T}{\partial x}\right|_{i+\frac{1}{2},j} = \frac{\bar{T}_{i+1,j}-\bar{T}_{i,j}}{\Delta x} + O\left(\Delta x^2\right)$$

and

$$\left.\frac{\partial T}{\partial y}\right|_{i,j+\frac{1}{2}} = \frac{\bar{T}_{i,j+1}-\bar{T}_{i,j}}{\Delta y} + O\left(\Delta y^2\right)$$

3.4 Flux Integrals

Equation 3.5 requires us to evaluate the integral of the normal flux around each control volume. That is, we need to compute $\oint_{\partial CV}\vec{F}\cdot\vec{n}\,dl$. For the control volume of Figure 3.1, we can write this as

$$\begin{aligned}\oint_{\partial CV}\vec{F}\cdot\vec{n}\,dl &= \vec{F}_{i+\frac{1}{2},j}\cdot\vec{n}_{i+\frac{1}{2},j}\Delta y + \vec{F}_{i,j+\frac{1}{2}}\cdot\vec{n}_{i,j+\frac{1}{2}}\Delta x \\ &\quad + \vec{F}_{i-\frac{1}{2},j}\cdot\vec{n}_{i-\frac{1}{2},j}\Delta y + \vec{F}_{i,j-\frac{1}{2}}\cdot\vec{n}_{i,j-\frac{1}{2}}\Delta x \\ &= \left(F_{x;i+\frac{1}{2},j} - F_{x;i-\frac{1}{2},j}\right)\Delta y + \left(F_{y;i,j+\frac{1}{2}} - F_{y;i,j-\frac{1}{2}}\right)\Delta x\end{aligned}$$

Returning once again to our Poisson example, we have to second-order accuracy

$$\begin{aligned}F_{x;i+\frac{1}{2},j} &= \frac{\bar{T}_{i+1,j}-\bar{T}_{i,j}}{\Delta x} \\ F_{x;i-\frac{1}{2},j} &= \frac{\bar{T}_{i,j}-\bar{T}_{i-1,j}}{\Delta x} \\ F_{x;i,j+\frac{1}{2}} &= \frac{\bar{T}_{i,j+1}-\bar{T}_{i,j}}{\Delta y}\end{aligned}$$

[2] You may also see this definition with the sign reversed; the difference is largely philosophical.

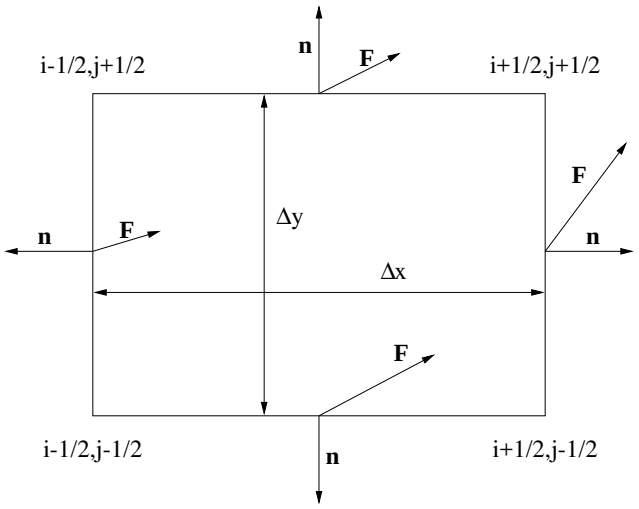


Figure 3.1: Flux integration around a finite volume.

$$\begin{aligned} F_{x;i,j-\frac{1}{2}} &= \frac{\bar{T}_{i,j} - \bar{T}_{i,j-1}}{\Delta y} \\ \oint_{\partial CV} \vec{F} \cdot \vec{n}\, dl &= \left(\bar{T}_{i+1,j} - 2\bar{T}_{i,j} + \bar{T}_{i-1,j}\right) \frac{\Delta y}{\Delta x} + \left(\bar{T}_{i,j+1} - 2\bar{T}_{i,j} + \bar{T}_{i,j-1}\right) \frac{\Delta x}{\Delta y} \end{aligned}$$

Substituting this into Equation 3.8 and dividing by $A = \Delta x \Delta y$, we get the canonical finite-volume discretization of Poisson's equation.

$$\frac{\bar{T}_{i+1,j} - 2\bar{T}_{i,j} + \bar{T}_{i-1,j}}{\Delta x^2} + \frac{\bar{T}_{i,j+1} - 2\bar{T}_{i,j} + \bar{T}_{i,j-1}}{\Delta y^2} = \bar{S} \tag{3.12}$$

It is easy to show by Taylor analysis that the left-hand side of Equation 3.12 is a second-order accurate approximation to the Laplacian of $\bar{T}$ at i, j.

3.5 Problems

1. Show that, for a smooth function, the difference between $T_i$and $\bar{T}_i$ is $O\left(\Delta x^2\right)$. (Hint: expand T in a Taylor series about $x = x_i$.

2. Show that

$$\overline{\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right)}_{i,j} = \frac{\bar{T}_{i+1,j} - 2\bar{T}_{i,j} + \bar{T}_{i-1,j}}{\Delta x^2} + \frac{\bar{T}_{i,j+1} - 2\bar{T}_{i,j} + \bar{T}_{i,j-1}}{\Delta y^2} + O\left(\Delta x^2, \Delta y^2\right)$$

3. **High-order accurate flux evaluation for Poisson's equation.** Suppose that we wanted a more accurate approximation for the flux for Poisson's equation than we got in Section 3.3. We could choose to use four control volume averages to compute the flux: $\overline{T}_{i+2}$, $\overline{T}_{i+1}$, $\overline{T}_i$, and $\overline{T}_{i-1}$. Find the most accurate possible approximation to the $\frac{\partial T}{\partial x}_{i+\frac{1}{2}}$ and determine the leading-order truncation error term. Combine this flux with its analog at $i-\frac{1}{2}$ to get a high-order approximation to the Laplacian in 1D, and find the truncation error for this Laplacian approximation.

4. Show that the flux for the control volume boundary at $i+\frac{1}{2}$ for the wave equation really is $T_{i+\frac{1}{2}}$.

5. **First-order upwind flux for the wave equation.** The flux $T_{i+\frac{1}{2}}$ can be approximated most simply by using data from the control volume upwind of the interface; for a positive wave speed, this is control volume i. Show that this approximation is only first-order accurate.

6. **Centered flux for the wave equation.** Suppose we were to use two control volume averages ($\bar{T}_i$ and $\bar{T}_{i+1}$) to evaluate the flux at $i+\frac{1}{2}$. Find an expression for the flux, determine the accuracy of the flux (including the leading-order term in the truncation error), and find the flux integral for the 1D case.

7. **Upwind extrapolated flux for the wave equation**. Suppose that we wanted a more accurate approximation for the flux for the wave equation while still using upwind data. We could choose to use two control volume averages to compute the flux at $i+\frac{1}{2}$: $\overline{T}_i$ and $\overline{T}_{i-1}$. Find the most accurate possible approximation to the flux and determine the leading-order truncation error term.

Chapter 4

Practical Aspects of Solving Poisson's Equation

4.1 Solving the Discrete Poisson Equation

Learning Objectives. Students will be able to:

- List four categories of techniques for solving Laplace's equation.
- Explain why direct solution of Laplace's equation is impractical.
- Describe how to implement five variants of point and line iterative methods.
- Explain why SOR and SLOR methods are more efficient than variants without over-relaxation.

Suppose we want to solve Poisson's equation on the unit square

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = S(x,y) \qquad (x,y)\,\varepsilon\,[0,1]\times[0,1]$$

We'll talk about boundary conditions later. In the interior of the domain, we can discretize the equation by

$$\frac{\bar{T}_{i+1,j} - 2\bar{T}_{i,j} + \bar{T}_{i-1,j}}{\Delta x^2} + \frac{\bar{T}_{i,j+1} - 2\bar{T}_{i,j} + \bar{T}_{i,j-1}}{\Delta y^2} = S_{i,j} \tag{4.1}$$

This discretization is second-order accurate in both x and y, which you can all easily verify at this point. If we write Equation 4.2 for every point in the mesh, we get a matrix equation of size $N \equiv i_{max} j_{max}$. In particular, for a 4-by-4 mesh with no source term, we get:

$$\begin{bmatrix}
D & X & & & Y & & & & & & & & & & & \\
X & D & X & & & Y & & & & & & & & & & \\
 & X & D & X & & & Y & & & & & & & & & \\
 & & X & D & & & & Y & & & & & & & & \\
Y & & & & D & X & & & Y & & & & & & & \\
 & Y & & & X & D & X & & & Y & & & & & & \\
 & & Y & & & X & D & X & & & Y & & & & & \\
 & & & Y & & & X & D & & & & Y & & & & \\
 & & & & Y & & & & D & X & & & Y & & & \\
 & & & & & Y & & & X & D & X & & & Y & & \\
 & & & & & & Y & & & X & D & X & & & Y & \\
 & & & & & & & Y & & & X & D & & & & Y \\
 & & & & & & & & Y & & & & D & X & & \\
 & & & & & & & & & Y & & & X & D & X & \\
 & & & & & & & & & & Y & & & X & D & X \\
 & & & & & & & & & & & Y & & & X & D
\end{bmatrix}
\begin{pmatrix}
\bar{T}_{1,1} \\ \bar{T}_{2,1} \\ \bar{T}_{3,1} \\ \bar{T}_{4,1} \\ \bar{T}_{1,2} \\ \bar{T}_{2,2} \\ \bar{T}_{3,2} \\ \bar{T}_{4,2} \\ \bar{T}_{1,3} \\ \bar{T}_{2,3} \\ \bar{T}_{3,3} \\ \bar{T}_{4,3} \\ \bar{T}_{1,4} \\ \bar{T}_{2,4} \\ \bar{T}_{3,4} \\ \bar{T}_{4,4}
\end{pmatrix} = 0 \tag{4.2}$$

where $X \equiv \frac{1}{\Delta x^2}, Y \equiv \frac{1}{\Delta y^2}$, and $D = -2(X+Y)$. Note that the diagonals with Y's in them are separated from the main diagonal (with the D's) by i_{max} entries.

There are at least four ways to solve this huge matrix equation:

1. Direct inversion via Gaussian elimination, LU decomposition, etc. This is expensive. Even using the bandedness of the matrix, we still require in general $O\left(i_{max}^3 j_{max}\right)$ operations for direct inversion. As mesh sizes increase, this soon becomes impractical.

2. Krylov subspace methods for solving linear systems seek to find a linear combination of vectors which minimize the residual $\vec{R} \equiv [A]\vec{x} - \vec{b}$ for a matrix equation; this linear combination is used to update $\vec{x}$. Examples of such solvers include GMRES and BiCGStab. These methods are fairly memory intensive and generally require a reasonable approximate inverse to the original matrix as a pre-conditioner. While these solvers are very effective, they are also quite complex, and we aren't going to discuss them.

3. Point and line iterative methods don't pretend to be able to get the right answer in a single pass. They act by updating the solution at one point or along one line in the mesh at a time. Many iterations are required to converge, but each iteration is very cheap. The drawback to many of these methods is that they are extremely bad at damping long-wavelength errors in the solution.

4. Multigrid methods are typically used in conjunction with simple iterative methods. An iterative method is used to damp the highest frequency errors. Then the remaining error is used to drive a Poisson problem on a mesh with half as many points in each direction. When this problem has been solved, perhaps by using multigrid recursively, a correction is interpolated back to the fine mesh. Both in theory and in practice, a well-designed multigrid method can solve Laplace's equation to machine zero in a computational cost equivalent to around ten applications of the iterative method used on the finest mesh.

All of these techniques can also be used to solve the systems of equations arising from discretization of the Navier-Stokes equations — and all of them have been used successfully. For our purposes in this course, we'll focus on point and line iterative methods.

4.1.1 Iterative Methods for Poisson's Equation

We've discretized Poisson's equation as

$$\frac{\bar{T}_{i+1,j} - 2\bar{T}_{i,j} + \bar{T}_{i-1,j}}{\Delta x^2} + \frac{\bar{T}_{i,j+1} - 2\bar{T}_{i,j} + \bar{T}_{i,j-1}}{\Delta y^2} = S_{i,j} \tag{4.3}$$

We can solve this for $\bar{T}_{i,j}$:

$$\bar{T}_{i,j}\frac{2\Delta x^2 + 2\Delta y^2}{\Delta x^2 \Delta y^2} = \frac{\bar{T}_{i+1,j} + \bar{T}_{i-1,j}}{\Delta x^2} + \frac{\bar{T}_{i,j+1} + \bar{T}_{i,j-1}}{\Delta y^2} - S_{i,j}$$

Using k as an index for the iteration number, the simplest choice we could make in an iteration scheme would be

Point Jacobi method

$$\bar{T}_{i,j}^{k+1} = \frac{\Delta y^2}{2\left(\Delta x^2 + \Delta y^2\right)}\left(\bar{T}_{i+1,j}^{k} + \bar{T}_{i-1,j}^{k}\right) + \frac{\Delta x^2}{2\left(\Delta x^2 + \Delta y^2\right)}\left(\bar{T}_{i,j+1}^{k} + \bar{T}_{i,j-1}^{k}\right) - S_{i,j}\frac{\Delta x^2 \Delta y^2}{2\left(\Delta x^2 + \Delta y^2\right)} \tag{4.4}$$

To compute all new values of $\bar{T}_{i,j}^{k+1}$, we would sweep through the entire mesh. This scheme requires storage for two copies of $\bar{T}$. We can both reduce storage and (it turns out) improve efficiency by using the latest available data while sweeping through the mesh. If we sweep in order of increasing i and then increasing j,[1] our iteration scheme would formally look like:

Point Gauss-Seidel method

$$\bar{T}_{i,j}^{k+1} = \frac{\Delta y^2}{2\left(\Delta x^2+\Delta y^2\right)}\left(\bar{T}_{i+1,j}^{k}+\bar{T}_{i-1,j}^{k+1}\right)+\frac{\Delta x^2}{2\left(\Delta x^2+\Delta y^2\right)}\left(\bar{T}_{i,j+1}^{k}+\bar{T}_{i,j-1}^{k+1}\right)-S_{i,j}\frac{\Delta x^2\Delta y^2}{2\left(\Delta x^2+\Delta y^2\right)} \tag{4.5}$$

This looks harder to code, but it isn't. With only one array to store $\bar{T}$, the data to compute $\bar{T}_{i,j}^{k+1}$ automatically comes from the iteration levels in Equation 4.5. The point Gauss-Seidel scheme can be shown both analytically and computationally to be more efficient than point Jacobi.

One thing that quickly becomes apparent when looking in detail at the computational behavior of these schemes is that the update to the solution is always (much) smaller that it needs to be. A logical thing to try, then, is to increase the update by some factor. When applied to point Gauss-Seidel, this results in the following scheme:

SOR

$$\begin{aligned}\delta T_{i,j}^{k+1} &= \frac{\Delta y^2}{2\left(\Delta x^2+\Delta y^2\right)}\left(\bar{T}_{i+1,j}^{k}+\bar{T}_{i-1,j}^{k+1}\right)+\frac{\Delta x^2}{2\left(\Delta x^2+\Delta y^2\right)}\left(\bar{T}_{i,j+1}^{k}+\bar{T}_{i,j-1}^{k+1}\right)-S_{i,j}\frac{\Delta x^2\Delta y^2}{2\left(\Delta x^2+\Delta y^2\right)}-\bar{T}_{i,j}^{k}\\ \bar{T}_{i,j}^{k+1} &= \bar{T}_{i,j}^{k}+\omega\delta T_{i,j}^{k+1}\end{aligned} \tag{4.6}$$

This scheme is referred to as successive over-relaxation (SOR); the over-relaxation parameter, ω, must be less than two for the iterative scheme to be stable.

Test case: Solve Laplace's equation with boundary conditions $T(x,0)=T(0,y)=0$, $T(x,1)=\sin(\pi x/2)$, and $T(1,y)=\sin(\pi y/2)$ and initial guess of 0. At this point, we aren't worried about the exact solution, but only about the rate of convergence. This is commonly measured by computing norms of the change in the solution from one iteration to the next. For point Jacobi, point Gauss-Seidel, and point Gauss-Seidel with SOR (ω = 1.8), the L_1 norm of the change in solution is shown in Figure 4.1. Notice that the point Jacobi scheme appears to converge faster initially

[1] That is, with loops like this:

```
for j = 1, jmax
  for i = 1, imax
    ...
  end for
end for
```

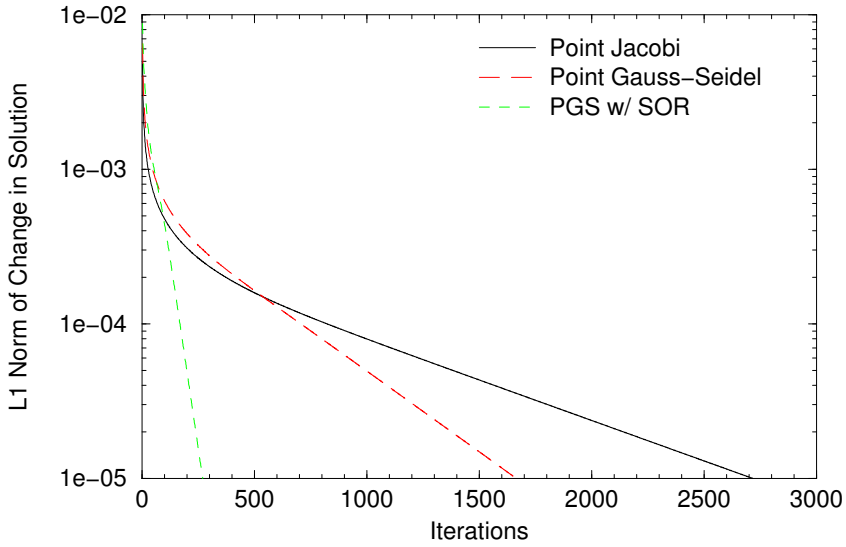


Figure 4.1: Comparison of convergence rates for point iterative schemes applied to Laplace's equation.

(say the first 50 iterations or so). This is an illusion. The point Jacobi scheme is so local in its effect that this initially-fast convergence merely reflects the slow rate at which information "propagates" via the iterative scheme. This corresponds to very poor damping rates for low-frequency (long wavelength) errors.

One can also solve simultaneously for all values along a line in the mesh. For example, we could obtain all the $\bar{T}_{i,j}^{k+1}$ with the same value of i by solving this equation:

Line Gauss-Seidel method

$$
\begin{aligned}
-\frac{\Delta x^2}{2\left(\Delta x^2+\Delta y^2\right)}\delta T_{i,j+1}^{k+1}+\delta T_{i,j}^{k+1} & \\
-\frac{\Delta x^2}{2\left(\Delta x^2+\Delta y^2\right)}\delta T_{i,j-1}^{k+1} &= \frac{\Delta x^2}{2\left(\Delta x^2+\Delta y^2\right)}\left(\bar{T}_{i,j+1}^{k}+\bar{T}_{i,j-1}^{k}\right) \qquad (4.7)\\
&\quad +\frac{\Delta y^2}{2\left(\Delta x^2+\Delta y^2\right)}\left(\bar{T}_{i+1,j}^{k}+\bar{T}_{i-1,j}^{k+1}\right)-S_{i,j}\frac{\Delta x^2\Delta y^2}{2\left(\Delta x^2+\Delta y^2\right)} \qquad (4.8)\\
\bar{T}_{i,j}^{k+1} &= \bar{T}_{i,j}^{k}+\omega\delta T_{i,j}^{k+1}
\end{aligned}
$$

This equation presumes that we are marching across lines in the order of increasing i, so that data at $i-1$ is available while we are solving along line i. Successive over-relaxation can be used with this scheme by setting $1<\omega<2$. Figure 4.2 compares

the convergence rates of line and point Gauss-Seidel iterative methods. The line methods clearly require fewer iterations. However, line iterations take about three times as long as point iterations, so the point methods are faster *for this problem*. In general, line iterative schemes are very effective when applied along a direction of strong coupling — for example, across the boundary layer in a viscous flow simulation. Where coupling is less strong (as in this case for the Laplace equation), line methods are less effective. Line methods also become relatively more efficient when the number of mesh points rises because information is "propagated" faster by the iterative scheme.

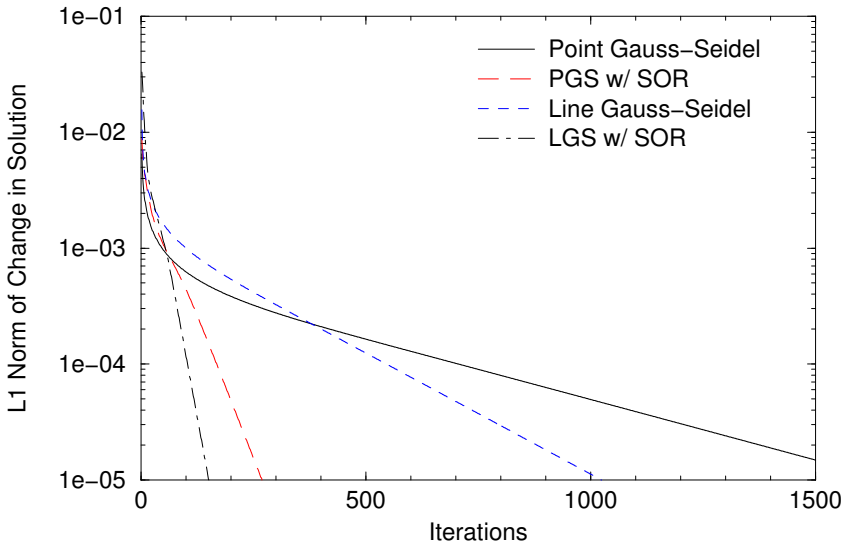


Figure 4.2: Comparison of convergence rates for point and line Gauss-Seidel iterative schemes applied to Laplace's equation.

4.2 Boundary Conditions for the Laplacian

Learning Objectives. Students will be able to:

- Describe how to implement Neumann, Dirichlet, and mixed boundary conditions for Poisson's equation in finite-volume form.

There are (at least) three categories of boundary conditions for PDE's: those that prescribe the solution on the boundary (Dirichlet conditions), those that prescribe

the gradient of the solution on the boundary (Neumann conditions), and those that prescribe some relationship between the solution and its gradient on the boundary (mixed conditions). When solving Laplace's equation for a temperature field, these correspond to fixed temperature, fixed heat flux, and convection or radiation boundary conditions respectively.

4.2.1 Neumann (Fixed Heat Flux) Boundary Condition

In the finite-volume formulation, we compute the integral of the flux around each control volume. This makes it trivial to impose Neumann boundary conditions. For example, in Figure 4.3, the flux integral is computed as usual, except that the flux along the $i, j - \frac{1}{2}$ side of the finite volume is replaced by the prescribed boundary flux — in this case, the most common value of zero is shown as an example.

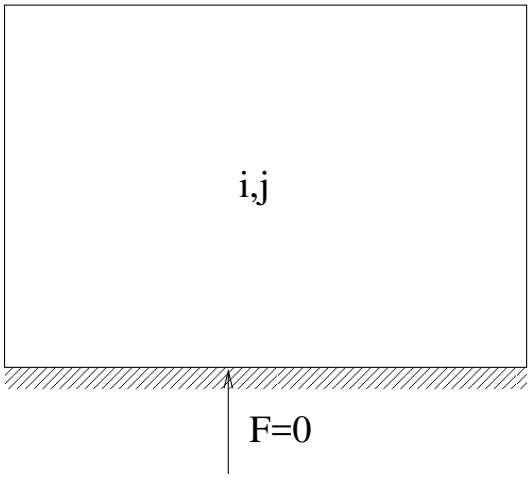


Figure 4.3: Finite volume with homogeneous Neumann boundary condition imposed along once side.

4.2.2 Dirichlet (Fixed Temperature) Boundary Condition

The situation is more complex when we are faced with a Dirichlet boundary condition, because we have no way of directly imposing a value on the solution at the boundary. We could set the value in cell i, j of Figure 4.3 to the given wall value, but this is physically incorrect. Why? Because the solution value stored for cell i, j is

the *average value* over that control volume. Imposing — for example — $T = 300$K at the wall is not at all the same as saying that the average value of temperature in the control volume next to the wall is 300K; any solution with a temperature gradient will be adversely affected by this incorrect boundary condition.

What we need to do for the control volumes next to the wall is to compute a physically correct flux at the wall. To do this, we need to compute a temperature gradient at the wall. There are two straightforward ways to do this; the results are identical mathematically, but they are programmed differently.

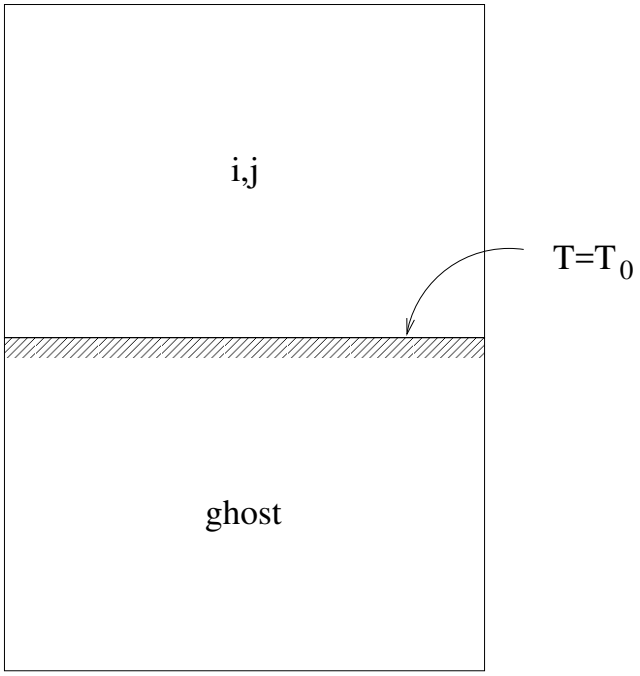


Figure 4.4: Boundary cell showing Dirichlet boundary condition and ghost cell.

Approach 1: One-sided differences

One choice for computing the flux at $i, j - \frac{1}{2}$ is to use one-sided differences. That is, compute

$$\left.\frac{\partial T}{\partial y}\right|_{i,j-\frac{1}{2}} = \frac{T_{i,j} - T_w}{\Delta y/2}$$

and use this flux when computing the flux integral.

Approach 2: Ghost cells

Another choice is to create a *ghost cell* at $i, j-1$. This ghost cell lies outside the computational domain, so any solution value we assign to it is purely fictitious except for its role in enforcing the boundary condition. If we set the temperature in this ghost cell by linear extrapolation using $T_{i,j}$ and T_w, we will get a temperature in the ghost cell of

$$T_{i,j-1} = 2T_w - T_{i,j}$$

Using this value to compute the flux at the boundary results in

$$\left.\frac{\partial T}{\partial y}\right|_{i,j-\frac{1}{2}} = \frac{T_{i,j} - T_{i,j-1}}{\Delta y} = \frac{T_{i,j} - 2T_w + T_{i,j}}{\Delta y} = \frac{T_{i,j} - T_w}{\Delta y/2}$$

The flux is the same.

The difference lies in how one chooses to program the boundary conditions. One-sided differencing requires a change in the way the flux is calculated for the boundary; ghost cells require an additional row of cells and some work to set values in these cells, but flux calculations are the same at the domain boundary as at interior control volume boundaries.

Chapter 5

Accuracy Assessment for Numerical Solutions

5.1 If an exact solution is available

Suppose that for some problem of interest we have an exact solution $u_e(x,y)$ and a numerical solution $\hat{u}_{\Delta x}(x_{i,j},y_{i,j})$ on a mesh with spacing Δx. The error $E_{i,j}$ in the numerical solution is:

$$E_{i,j;\Delta x} = u_e\left(x_{i,j},y_{i,j}\right) - \hat{u}_{\Delta x}\left(x_{i,j},y_{i,j}\right)$$

So that's simple enough, and so is plotting the error. This can give useful information about the location and (often) the source of numerical errors. It can also give useful information about places where the solution is not resolved well enough; poor resolution leads to increased truncation error, which will show up in these plots.

To summarize the error as a single number, there are three commonly-used norms:

$$\left\|E_{i,j}\right\|_1 = \frac{\sum_i \sum_j \left|E_{i,j}\right|}{i_{max} j_{max}} \tag{5.1}$$

$$\left\|E_{i,j}\right\|_2 = \sqrt{\frac{\sum_i \sum_j E_{i,j}^2}{i_{max} j_{max}}} \tag{5.2}$$

$$\left\|E_{i,j}\right\|_\infty = \max_{i,j} \left|E_{i,j}\right| \tag{5.3}$$

Order of accuracy can be determined by computing some error norm for each of a series of meshes and determining the slope on a log-log plot of the error versus Δx, for example. This slope will be the order of accuracy of the method. In general, the order of accuracy determined in this way will not be exactly 1, 2, 3, etc. Variations of as much as 0.2 or so are routinely accepted as insignificant in this sort of analysis.

The global norms L_1(Equation. 5.1) and L_2(Equation. 5.2) often converge a half or full order faster than the L_∞ norm (Equation. 5.3), which is a local measure. That is, the L_∞ norm can converge as $O(\Delta x)$ because of a local error at a point, while the L_2 norm will converge as $O\left(\Delta x^{3/2}\right)$ and the L_1 norm will converge as $O\left(\Delta x^2\right)$. While this is not *always* true, it does happen sometimes.

5.2 If an exact solution is *not* available

Suppose we have solutions on three meshes M_1, M_2, and M_3, where M_2 has twice as many mesh points as M_1, and M_3 has twice as many as M_2. We assume that the error in each solution is proportional to its mesh spacing to some power k; then we can write the solutions as:

$$u|_{M_1} = u_e + C\Delta x^k$$

$$u|_{M_2} = u_e + C\left(\frac{\Delta x}{2}\right)^k$$

$$u|_{M_3} = u_e + C\left(\frac{\Delta x}{4}\right)^k$$

Taking the norm of the difference of the solutions, we expect to get:

$$\|u|_{M_1} - u|_{M_2}\| = C\Delta x^k\left(1 - \frac{1}{2^k}\right)$$

and

$$\|u|_{M_2} - u|_{M_3}\| = C\Delta x^k\left(\frac{1}{2^k} - \frac{1}{4^k}\right)$$

First, the difference should get smaller as we refine the mesh. Second, if we take the ratio of these last two expressions, we get

$$\frac{\|u|_{M_2} - u|_{M_3}\|}{\|u|_{M_1} - u|_{M_2}\|} = \frac{\frac{1}{2^k}\left(1 - \frac{1}{2^k}\right)}{\left(1 - \frac{1}{2^k}\right)} = \frac{1}{2^k}$$

Clearly, we can use this to evaluate k. And there's more good news: we can estimate the error norm for the finest-mesh solution. That norm error is $C\Delta x^k \frac{1}{4^k}$. The norm difference between solutions on M_2 and M_3 is $C\Delta x^k \left(\frac{1}{2^k} - \frac{1}{4^k}\right)$. The ratio of these two is:

$$\frac{\|E_{M_3}\|}{\|u|_{M_2} - u|_{M_3}\|} = \frac{\frac{1}{4^k}}{\frac{2^k-1}{4^k}} = \frac{1}{2^k - 1}$$

So now we can estimate the error norm for the finest mesh.

This approach has several pitfalls.

- The solution must be continuous, because otherwise error norms are very tricky to evaluate.
- Each of the three solutions must be accurate enough (features must be well-enough resolved) that the error may be assumed to follow its asymptotic behavior.
- In any event, the error norm that is computed is not the most reliable estimate in the world.

5.3 Problems

1. For a particular discrete problem, the L_2-norm of the error in the solution (measured by comparison with a known exact solution) is given by:

Mesh	L_2	Ratio
10×10	$4.68 \cdot 10^{-2}$	—
20×20	$9.08 \cdot 10^{-3}$	5.15
40×40	$2.13 \cdot 10^{-3}$	4.26
80×80	$5.32 \cdot 10^{-4}$	4.00

   What is going on here? What would you estimate is the true order of accuracy?

2. Suppose that you are solving a problem for which you do not have an analytic comparison solution. You take norms of the difference in solutions on different meshes and get the following data:

Mesh 1	Mesh 2	L_2
20×20	40×40	$1.25 \cdot 10^{-3}$
40×40	80×80	$1.78 \cdot 10^{-4}$
80×80	160×160	$2.55 \cdot 10^{-5}$

   Find the actual numerical order of accuracy of the scheme and estimate the error in the computed solution on the finest mesh. What do you think is the order of accuracy that the scheme is analytically expected to achieve?

3. For unstructured meshes, estimating order of accuracy is complicated somewhat because one can't just double the number of cells in each direction. The following table contains error data for a 2-D advection-diffusion problem (as calculated by my research code), using an exact solution for comparison. Estimate the order of accuracy for each norm.

# cells	L_1	L_2	L_∞
64	$3.923 \cdot 10^{-3}$	$4.669 \cdot 10^{-3}$	$9.370 \cdot 10^{-3}$
240	$1.295 \cdot 10^{-3}$	$1.716 \cdot 10^{-3}$	$6.239 \cdot 10^{-3}$
922	$1.965 \cdot 10^{-4}$	$2.656 \cdot 10^{-4}$	$1.721 \cdot 10^{-3}$

Chapter 6

Time Accuracy and Stability Analysis for Ordinary Differential Equations

As we shall see, the space discretization of a partial differential equation in one space dimension results in a coupled system of ordinary differential equations in time, one equation for each unknown in the spatial mesh. It is possible to analytically transform this system of ODE's into an equivalent decoupled system. While there is no practical application for this transformation in terms of how we solve a system of PDE's, the decoupled system is much easier to analyze to determine the time accuracy and stability properties of a numerical scheme.

Accompanying this theoretical discussion is a set of examples showing how to apply these techniques to real time advance schemes.

6.1 From PDE to Coupled ODE's

Learning Objectives. Students will be able to:

- Convert the spatially-discretized form of a time-dependent PDE with periodic boundary conditions into a system of coupled ODE's.

Suppose that we have a generic space discretization for a PDE in x and t written as

$$\left.\frac{\partial T}{\partial t}\right|_i \equiv \frac{dT_i}{dt} = a_{-2}T_{i-2} + a_{-1}T_{i-1} + a_0 T_i + a_1 T_{i+1} + a_2 T_{i+2}$$

This is referred to as the *semi-discrete form* of the PDE, because the equation has been discretized in space but not in time. Now let's write the semi-discrete form of the equation for every point in the mesh, assuming *periodic boundary conditions* (much as we would have if we were solving on the surface of a cylinder, for instance). This gives us a coupled set of ODE's for the T_i.

$$\begin{aligned}
\frac{dT_0}{dt} &= a_{-2}T_{i_{\max}-2} + a_{-1}T_{i_{\max}-1} + a_0T_0 + a_1T_1 + a_2T_2 \\
\frac{dT_1}{dt} &= a_{-2}T_{i_{\max}-1} + a_{-1}T_0 + a_0T_1 + a_1T_2 + a_2T_3 \\
\frac{dT_2}{dt} &= a_{-2}T_0 + a_{-1}T_1 + a_0T_2 + a_1T_3 + a_2T_4 \\
&\vdots \\
\frac{dT_i}{dt} &= a_{-2}T_{i-2} + a_{-1}T_{i-1} + a_0T_i + a_1T_{i+1} + a_2T_{i+2} \\
&\vdots \\
\frac{dT_{i\max-1}}{dt} &= a_{-2}T_{i_{\max}-3} + a_{-1}T_{i_{\max}-2} + a_0T_{i_{\max}-1} + a_1T_0 + a_2T_1
\end{aligned}$$

This can be re-written as:

$$\frac{d}{dt}\begin{pmatrix} T_0 \\ T_1 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_{i\max-1} \end{pmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 & & & a_{-2} & a_{-1} \\ a_{-1} & a_0 & a_1 & a_2 & & & a_{-2} \\ a_{-2} & a_{-1} & a_0 & a_1 & a_2 & & \\ & & & \ddots & & & \\ & & a_{-2} & a_{-1} & a_0 & a_1 & a_2 \\ & & & & & \ddots & \\ a_1 & a_2 & & & a_{-2} & a_{-1} & a_0 \end{bmatrix} \begin{pmatrix} T_0 \\ T_1 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_{i\max-1} \end{pmatrix}$$

or as

$$\frac{d\vec{T}}{dt} = B_p(a_{-2}, a_{-1}, a_0, a_1, a_2)\,\vec{T} \tag{6.1}$$

where B_p is compact notation for the banded periodic matrix written above. By convention, we will pad the list of coefficients with zeroes to ensure that a_0 is always the middle entry in the list.

So far, nothing fancy has happened — we've just discretized the PDE in space and manipulated the result into a convenient form. This approach will *always* work; no matter what differential operator we have in space or what discretization we use for it, an equation like 6.1 can always be derived, so long as we use the same spatial discretization for every control volume. The only difference among such equations is the number of diagonals in the *banded periodic matrix* and what numbers go into each diagonal.

It can be shown (see Appendix B) that the matrix $B_p(a_{-2}, a_{-1}, a_0, a_1, a_2)$ has a *complete eigensystem*. Therefore, we can construct a matrix X whose columns are the right eigenvectors of B_p and use it to diagonalize the system in Equation 6.1:

$$\begin{aligned} X^{-1}\frac{d\vec{T}}{dt} &= X^{-1}B_p\left(XX^{-1}\right)\vec{T} \\ \frac{d\left(X^{-1}\vec{T}\right)}{dt} &= \left(X^{-1}B_pX\right)\left(X^{-1}\vec{T}\right) \\ \frac{d\vec{w}}{dt} &= \Lambda\vec{w} \end{aligned}$$

where $\vec{w} \equiv X^{-1}\vec{u}$ is a new set of unknowns and Λ is a diagonal matrix whose diagonal entries are the eigenvalues of B_p. This is a system of $i_{\max}$ *uncoupled* ODE's. Solving this system is equivalent to solving Equation 6.1.

Summary We began with a PDE; discretized it in space to get a system of coupled ODE's; and diagonalized that system to get an uncoupled system of ODE's.[1] Because the two systems of ODE's are completely equivalent, the stability limitations for a time advance method applied to each of them is the same. This means that *we can analyze the stability of time advance methods completely independently from space discretization methods.*

Analysis of a time advance scheme for a model ODE will tell us what eigenvalues the matrix B_p can have for the combined space and time discretization scheme to be stable. In fact, we can easily get a bit more than that: we can find the amplification factor σ for any eigenvalue in the complex plane. This information is independent

[1] Note that this isn't typically useful in a practical sense, because we rarely have periodic boundary conditions and often are solving non-linear equations. So we don't solve real problems using this transformation; we just use the transformation to help us analyze time advance schemes.

of the spatial scheme that produced the eigenvalue; the time advance analysis needs no information about the spatial discretization, not even the differential operator.

6.2 Analysis of Time March Schemes for ODE's

Learning Objectives. Students will be able to:

- Analyze the time accuracy of a time discretization for a model ODE.
- Determine the stability of a time discretization for a model ODE as a function of the eigenvalue λ appearing in the model ODE.
- Define amplification factor and describe how the amplification factor is related to stability for time-marching schemes.

We're going to analyze time advance schemes using the model ODE

$$\frac{dw}{dt} = \lambda w \tag{6.2}$$

The exact solution of this equation is:

$$w(t) = Ae^{\lambda t} \tag{6.3}$$

Consequently, v grows exponentially in time when the real part of λ is positive,$\Re(\lambda) > 0$ ($\equiv$inherently unstable); decays exponentially when $\Re(\lambda) < 0$ ($\equiv$inherently stable); and has constant amplitude when $\Re(\lambda) = 0$ ($\equiv$neutrally stable). The *amplification factor* σ of any solution — exact or numerical — is defined as the growth rate of the solution from time level $t = n\Delta t$ to time level $t + \Delta t = (n+1)\Delta t$. The amplification factor for the exact solution to the model ODE 6.2 is given by

$$\sigma_{\text{exact}} \equiv \frac{w(t+\Delta t)}{w(t)} \equiv \frac{w^{n+1}}{w^n} = e^{\lambda \Delta t} = 1 + \lambda\Delta t + \frac{(\lambda\Delta t)^2}{2} + \frac{(\lambda\Delta t)^3}{6} + \cdots \tag{6.4}$$

The difference between one numerical time advance scheme and another comes down to how we approximate the derivative on the left-hand side and the solution data on the right-hand side of Equation 6.2. We'll examine a number of alternatives.

In each case, we'll replace $\frac{dw}{dt}$ and λw with terms containing w^n, w^{n+1}, etc. Then we'll solve for the amplification factor $\sigma \equiv w^{n+1}/w^n$.

The ***accuracy*** of a time advance scheme depends on how well its amplification factor matches $e^{\lambda \Delta t}$ for small values of $\lambda \Delta t$; that is, on how many terms of the Taylor series expansion of Equation 6.4 are matched by the discrete scheme: the order of accuracy is equal to the exponent of the highest-order term that *matches* the exact amplification factor.[2] A time advance scheme is said to be ***stable*** for all complex eigenvalues λ for which the magnitude of the complex amplification factor $|\sigma| \leq 1$.

We can combine the analysis results for a spatial scheme (the eigenvalues λ as a function of the spatial step Δx) with the results for a time scheme (the amplification factor σ as a function of the eigenvalues λ and time step Δt) to determine the stability properties of a particular space/time discretization (the amplification factor σ as a function of Δx and Δt). This result will tell us whether there is a maximum stable time step for a given scheme, and if so what it is. See Chapter 6.5 for more information about this.

6.3 Caveats

Learning Objectives. Students will be able to:

- Explain two limitations of semi-discrete analysis for the Navier-Stokes equations.

There are several assumptions made in this analysis that should be explicitly stated, as they imply restrictions on the applicability of the results of the analysis.

Periodic boundary conditions. This analysis only applies to periodic boundary conditions. Similar but more complex analysis is possible to determine the eigenvalue structure for cases with more realistic boundary conditions; these eigenvalues can be used either analytically or graphically to show stability.

[2] Yes, this is slightly different than for space schemes, because of a difference in analysis approach. Using Taylor series expansions for time analysis gives results that are interpreted in the same way as Taylor analysis for space schemes.

Linearity and stationarity. We have assumed that the entries in B_p do not depend on the solution and do not change with time. That is, we assumed that the problem is *linear* and *stationary*. We can't guarantee that our results will have any meaning for non-linear or non-stationary problems — like the Navier-Stokes equations, for example.

Despite these restrictions, one can often perform this linear, periodic stability analysis and use the results to choose a time step for more complicated problems; generally, the maximum time step will have to be reduced by a factor of 0.6–0.8.

6.4 Examples

The first series of examples looks at eigenvalues for space discretization schemes.

6.4.1 Eigenvalues for the second-order accurate Laplacian operator

If we use the centered derivatives of Section 3.3 in a spatial discretization of the heat equation, we get

$$\frac{dT}{dt}_i = \alpha \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2} \tag{6.5}$$

so $B_p = B_p(0, \frac{\alpha}{\Delta x^2}, -\frac{2\alpha}{\Delta x^2}, \frac{\alpha}{\Delta x^2}, 0)$ and the eigenvalues are[3]:

$$\lambda = -\frac{2\alpha}{\Delta x^2}(1 - \cos\phi_k).$$

These eigenvalues fall on the negative real axis, between 0 and $-\frac{4\alpha}{\Delta x^2}$.

6.4.2 First-order upwind flux for the wave equation

In this case (see Section 3.5), the flux $T_{i+\frac{1}{2}}$ is approximated to first-order accuracy by using $\bar{T}_i$. To find the eigenvalues of this spatial operator, we first write the semi-discrete form of the governing equation by using the flux integral:

$$\frac{\partial \overline{T_i}}{\partial t} = -u \frac{\overline{T_i} - \overline{T_{i-1}}}{\Delta x}$$

[3] See Appendix B

$$= B_p\left(\frac{u}{\Delta x}, -\frac{u}{\Delta x}, 0\right)$$

Then the eigenvalues can be written down immediately:

$$\begin{aligned}\lambda_k &= \frac{u}{\Delta x}\left(e^{-I\phi_k} - 1\right) \\ &= \frac{u}{\Delta x}\left(-1 + \cos\phi_k - I\sin\phi_k\right)\end{aligned}$$

This is a circle of radius $\frac{u}{\Delta x}$ centered at $\left(-\frac{u}{\Delta x}, 0\right)$.

In the following series of examples, each time advance scheme is analyzed by using the model ODE defined in Equation 6.2 to determine both the accuracy and stability of the scheme.

6.4.3 Explicit Euler scheme

The explicit Euler time advance scheme uses *known* solution data at the current time level n to approximate the time derivative of the solution. When applied to the model ODE (Equation 6.2), this gives:

$$\frac{w^{n+1} - w^n}{\Delta t} = \lambda w^n \tag{6.6}$$

or

$$w^{n+1} = w^n(1 + \lambda\Delta t)$$

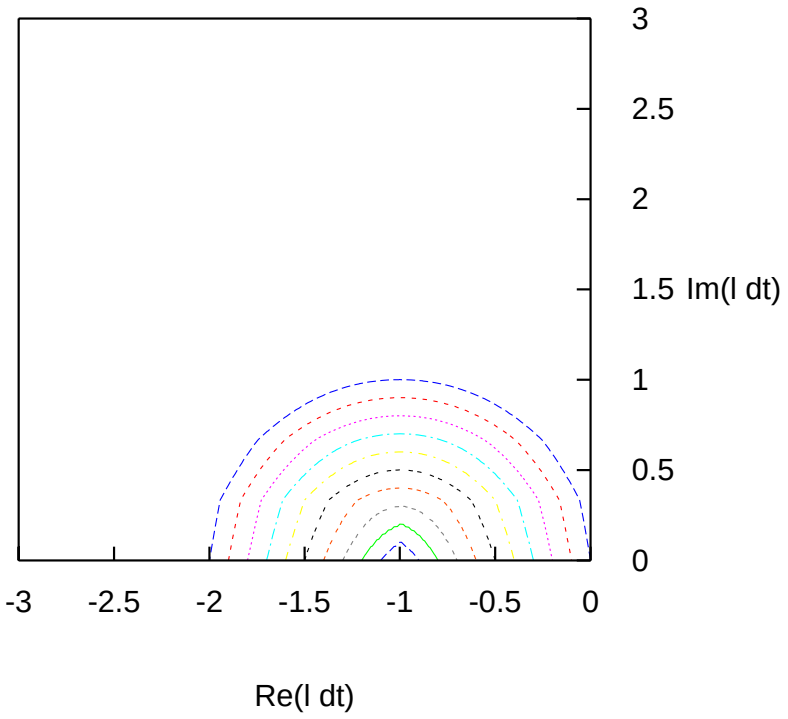
which implies that

$$\sigma = 1 + \lambda\Delta t$$

This scheme matches only the first-order term in the Taylor series expansion of $e^{\lambda\Delta t}$ and so is only first-order accurate. Regarding stability,

$$|\sigma| = \sqrt{(\Re(\lambda)\Delta t + 1)^2 + (\Im(\lambda)\Delta t)^2}$$

The contours of the magnitude of the amplification factor (for $|\sigma| \leq 1$) as a function of complex $\lambda\Delta t$ (written as $l\,dt$ in the captions) for this time advance scheme are:



Note that only the upper half of the complex plane is shown; the contours in the lower half of the plane are mirror images.

6.4.4 Implicit Euler scheme

The implicit Euler time advance scheme uses the *unknown* solution data at the $n+1$ time level to approximate the time derivative of the solution. When applied to the model ODE (Equation 6.2), this gives:

$$\frac{w^{n+1} - w^n}{\Delta t} = \lambda w^{n+1} \tag{6.7}$$

or

$$w^{n+1}(1 - \lambda \Delta t) = w^n$$

which implies that

$$\sigma = \frac{1}{1 - \lambda \Delta t}$$

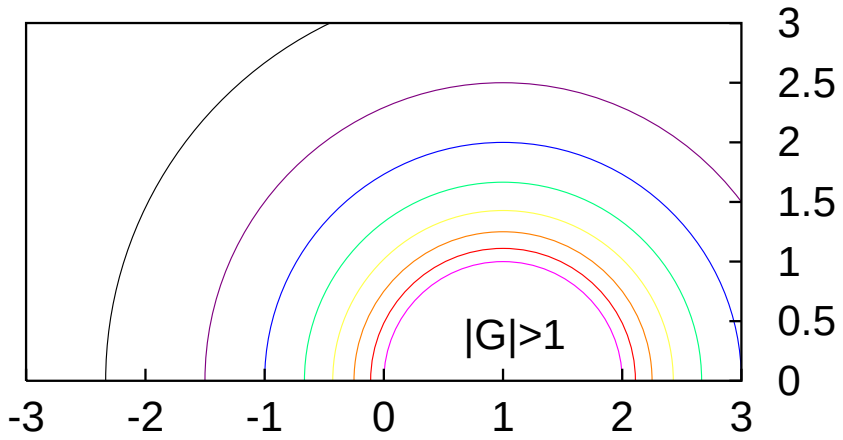
To determine the order of accuracy of this scheme, we need to be able to compare this amplification factor to the exact amplification factor of Equation 6.4. To do this, we note that (for $\lambda\Delta t < 1$)[4],

$$\sigma = \frac{1}{1-\lambda\Delta t} = 1 + \lambda\Delta t + (\lambda\Delta t)^2 + (\lambda\Delta t)^3 + \cdots$$

This scheme matches only the first-order term in the Taylor series expansion of $e^{\lambda\Delta t}$ and so is only first-order accurate. The magnitude of the amplification factor is given by:

$$\begin{aligned} |\sigma| &= \frac{1}{|1-\lambda\Delta t|} \\ &= \frac{1}{\sqrt{(1-\Re(\lambda)\Delta t)^2 + (\Im(\lambda)\Delta t)^2}} \end{aligned}$$

The contours of the amplification factor in the complex plane for this time advance scheme are shown below. It is easy to show that these contours are circles centered at $\lambda\Delta t = 1$. The scheme is unstable only for eigenvalues that fall inside a unit circle centered at this point.



[4]One can construct this using Taylor series. It's even easier to verify by multiplying the RHS by $1-\lambda\Delta t$.

6.4.5 Explicit Runge-Kutta schemes

Explicit Runge-Kutta time advance schemes are those schemes that have the following properties:

Explicit. The flux is evaluated from known quantities; no solution of linear systems is required.

Self-starting. No data from time level $n-1$ is required. A consequence of this property is that the amplification factor is single-valued.

Derivative-free. There is no need to approximate $\frac{dw}{dt}$ directly.

We have already encountered an explicit Runge-Kutta scheme: the explicit Euler scheme. As you will recall, when written for the model ODE, this scheme is

$$\frac{w^{n+1}-w^n}{\Delta t} = \lambda w^n$$

Clearly the properties of Runge-Kutta schemes are satisfied.

We will also use two- and four-stage Runge-Kutta schemes. There are numerous variants on these schemes, but the two we will use are:

$$\begin{aligned} w^{n+1} &= w^n + \lambda\,\Delta t\, w^{(1)} \\ w^{(1)} &= w^n + \frac{\lambda}{2}\Delta t\, w^n \end{aligned} \tag{6.8}$$

and

$$\begin{aligned} w^{n+1} &= w^n + \frac{\lambda\,\Delta t}{6}\left(w^n + 2w^{(1)} + 2w^{(2)} + w^{(3)}\right) \\ w^{(3)} &= w^n + \lambda\,\Delta t\, w^{(2)} \\ w^{(2)} &= w^n + \frac{\lambda}{2}\,\Delta t\, w^{(1)} \\ w^{(1)} &= w^n + \frac{\lambda}{2}\,\Delta t\, w^n \end{aligned} \tag{6.9}$$

A second-order Runge-Kutta scheme

This scheme is written for the model ODE as

$$\begin{aligned} w^{n+1} &= w^n + \lambda\,\Delta t\, w^{(1)} \\ w^{(1)} &= w^n + \frac{\lambda}{2}\Delta t\, w^n \end{aligned}$$

Combining these two, we get

$$\begin{aligned} w^{n+1} &= w^n + \lambda\Delta t\, w^n + \frac{(\lambda\Delta t)^2}{2} w^n \\ \sigma &= 1 + \lambda\Delta t + \frac{(\lambda\Delta t)^2}{2} \end{aligned}$$

So the scheme is evidently second-order accurate.

Now let's consider stability. Along the negative real axis, the amplification factor is:

$$\sigma = 1 + a + \frac{a^2}{2}$$

where $a \equiv \lambda\Delta t$ is real and negative. For $|\sigma| = 1$, we require:

$$a + \frac{a^2}{2} = 0$$

or $a = 0, -2$. The scheme is stable inside this region and unstable outside. This is easy to verify by picking any point where we haven't already shown $|\sigma| = 1$; if you want to be absolutely certain of your results, check one point in each region.

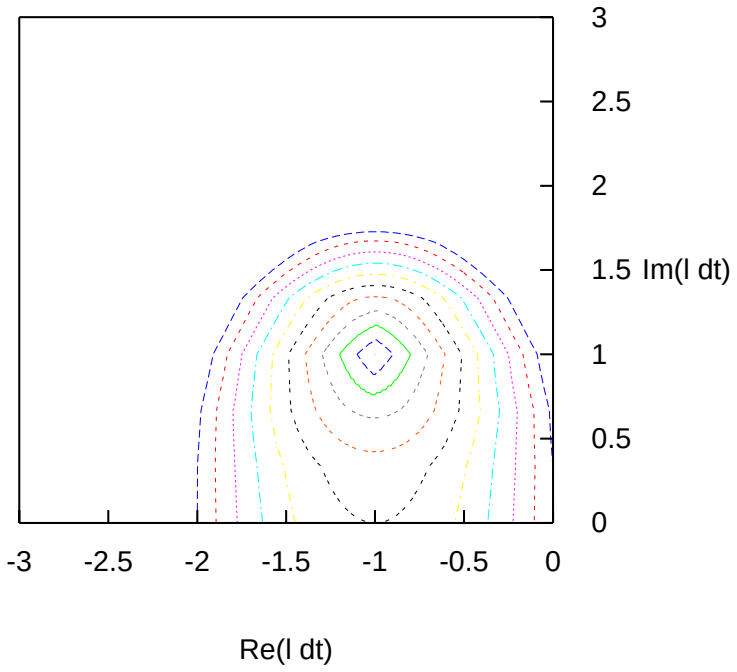
Along the imaginary axis,

$$\sigma = 1 + Ib - \frac{b^2}{2}$$

where $Ib \equiv \lambda\Delta t$ with b real. The magnitude of the amplification factor is

$$|\sigma| = \sqrt{1 - b^2 + \frac{b^4}{4} + b^2} = \sqrt{1 + \frac{b^4}{4}} > 1$$

The scheme is always unstable for pure imaginary eigenvalues. The contours of amplification factor in the complex λ plane for this scheme are shown below.



6.5 Stability Analysis for Fully-Discrete Systems

Learning Objectives. Students will be able to:

- Analyze the stability of a space-time discretization by combining the expression for the eigenvalues of the spatial operator with the amplification factor of the time discretization as a function of the eigenvalue.

We've seen how to analyze time discretization schemes and space discretization schemes in isolation. Analyzing time and space discretizations independently gives us full information about accuracy, and the two analyses provide complementary information about stability. Specifically, analysis of time discretization schemes applied to the model ODE tells us what the stability bounds are for a given time advance scheme, by providing an expression for amplification factor as a function of the eigenvalues of the space differencing scheme. Analysis of the space differencing scheme tells us what those eigenvalues are.

We can determine the stability of a fully-discrete approximation to a PDE A by combining the time and space analyses either analytically or graphically.

Analytically, we would substitute the eigenvalues of the spatial operator in the form $\lambda_k(\Delta x, \phi_k)$ into the amplification factor $\sigma(\lambda_k \Delta t)$. The stability limit for the combined scheme is determined by the maximum time step for which the worst-case amplification factor magnitude — that is, $\max_k |\sigma(\lambda_k(\Delta x, \phi_k)\Delta t)|$ — is less than one.

Graphically, we would plot the stability limits for the time discretization in the complex $\lambda\Delta t$ plane, then overlay curves showing $\lambda_k \Delta t(\Delta x, \phi_k)$ for the space differencing scheme for various non-dimensional times. Although less precise than analytic methods, the graphical approach is generally easier.

6.6 Examples

6.6.1 Upwind flux with explicit Euler time advance

We know from Section 3.5 that the eigenvalues for this spatial operator are:

$$\begin{aligned} \lambda_k &= \frac{u}{\Delta x}\left(e^{I\phi_k} - 1\right) \\ &= \frac{u}{\Delta x}(-1 + \cos\phi_k - I\sin\phi_k) \end{aligned}$$

This is a circle of radius $\frac{u}{\Delta x}$ centered at $\left(-\frac{u}{\Delta x}, 0\right)$.

Also, we know from Section 6.4.3 that the amplification factor for the explicit Euler time advance scheme is:

$$|\sigma| = \sqrt{(\Re(\lambda)\Delta t + 1)^2 + (\Im(\lambda)\Delta t)^2}$$

This quantity is less than one for any $\lambda\Delta t$ that falls within a circle of radius 1 centered at $(-1, 0)$. Therefore, the scheme is stable ($|\sigma| < 1$) if and only if $\frac{u\Delta t}{\Delta x} < 1$. The parameter $\frac{u\Delta t}{\Delta x}$ is called the *CFL number*, after three guys named Courant, Friedrichs, and Levy.

6.6.2 Centered flux with explicit Euler time advance

Using a result from Problem 6.2, we know the eigenvalues of this spatial operator:

$$\begin{aligned}\lambda_k &= \frac{u}{2\Delta x}\left(e^{I\phi_k} - e^{-I\phi_k}\right) \\ &= \frac{u}{\Delta x} I \sin\phi_k\end{aligned}$$

These eigenvalues lie on the imaginary axis, and therefore outside the stability range of the Euler time advance scheme (see Section 6.4.3) for any time step.

6.7 Problems

1. **Eigenvalues for the fourth-order accurate Laplacian operator.**

   Suppose that we use the fourth-order accurate Laplacian flux analyzed earlier in class to discretize the heat equation.[5] This gives us

   $$\frac{\partial \bar{T}}{\partial x}_i = \alpha \frac{-\bar{T}_{i+2} + 12T_{i+1} - 22\bar{T}_i + 12\bar{T}_{i-1} - \bar{T}_{i-2}}{8\Delta x^2}$$

   Find the banded periodic matrix associated with using this discretization on a periodic mesh, and find the eigenvalues of that matrix. Do these eigenvalues fall into the same range as those for the second-order accurate Laplacian space discretization?

2. **Eigenvalues for centered flux for the wave equation** Suppose that we choose to evaluate the flux for the wave equation by using $T_{i+\frac{1}{2}} \approx \left(\frac{\overline{T_i}+\overline{T_{i+1}}}{2}\right)$; this is second-order accurate. Write the semi-discrete form of the wave equation using this flux approximation and find the eigenvalues associated with the space scheme.

3. **Upwind Extrapolated Flux for the Wave Equation** If we use two control volumes upstream of an interface to estimate the flux for the wave equation,

[5] Note that this discretization is in terms of the *average* values in the control volumes. A finite-difference discretization, which assumes point-wise values of the solution at mesh points would give a different discretization. The reasons for the difference are somewhat technical, but hinge on the fact that the finite-volume flux implicitly assumes a cubic variation in the solution. This assumption has specific implications for the difference between the average value in the control volume $\bar{T}_i$ and the value at the center of cell i, $T(x_i)$, which in turn account for the difference between the finite difference and finite volume discretizations.

we get

$$T_{i+\frac{1}{2}} \approx \frac{3\bar{T}_i - \bar{T}_{i-1}}{2}$$

which is second-order accurate. Write the semi-discrete form of the wave equation using this flux approximation and find the eigenvalues associated with the space scheme. Plot these eigenvalues in the complex plane (remove the factor of $\frac{u}{\Delta x}$ before plotting).

4. **Trapezoidal scheme**. The trapezoidal scheme approximates the model ODE using a centered approximation for the data on the RHS:

$$\frac{w^{n+1} - w^n}{\Delta t} = \lambda \frac{w^{n+1} + w^n}{2}$$

Find the amplification factor for this scheme, and plot it. Where in the complex plane is the scheme stable?

5. **Fourth-order accurate Runge-Kutta scheme.** Analyze the fourth-order Runge-Kutta scheme given in Equation 6.9. Find and plot the amplification factor. Determine which region in your plot is the one where the scheme is stable.

6. **Extrapolated upwind flux with two-stage Runge-Kutta time advance**. The space scheme of Problem 3.7 and Problem 6.3 is second-order accurate in space, and will give a fully-discrete scheme that is second-order accurate when used in combination with the second-order Runge-Kutta time advance scheme. Combine the eigenvalues for this scheme (either graphically or analytically) with the amplification factor for the two-stage Runge-Kutta scheme of Section 6.4.5 to find the time step limit for the combined scheme.

7. **Heat equation, second-order in space, explicit Euler in time**. Suppose we wanted to use the second-order accurate Laplacian discretization in space and the explicit Euler time advance scheme (Section 6.4.3) to solve the heat equation. Using results from Section 6.4.1 and 6.4.3, find the amplification factor for this fully-discrete scheme and determine the maximum stable time step.

8. **Heat equation, fourth-order accurate in space, implicit Euler in time.** Repeat Problem 6.7 using the fourth-order accurate spatial discretization of Problems 3.3 and 6.1 and the implicit Euler time advance scheme.

Chapter 7

The Wave Equation

In previous chapters, we have discussed several space and time discretization schemes for the wave equation. In this chapter, we will look at the last issue remaining for the wave equation: boundary conditions. Then we will look at some sample solution for the wave equation using simple schemes, and finally explore some more advanced schemes for the wave equation that are more successful in the face of real-world complications.

The one-dimensional wave equation, $\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} = 0$, is very different from the Poisson and energy equations. Specifically,

- The wave equation is a hyperbolic PDE, whereas Poisson's equation is elliptic and the energy equation is parabolic. (See Section B.1 for an explanation of these terms.)
- The wave equation has fluxes that depend on the *solution*, whereas Poisson's equation has fluxes that depend on the *gradient of the solution*, and the energy equation has fluxes of both types.
- The wave equation and the energy equation are both time-dependent, but the wave equation, as we shall see, has a much less severe time step restriction than the energy equation.

Learning Objectives. Students will be able to:

- Describe two simple space-time discretizations for the one-dimensional wave equation and one major shortcoming of each.

- Explain why numerical schemes should (physically) use upwind data for the wave equation.
- Describe a second-order accurate upwind flux approximation for the wave equation and derive the eigenvalues obtained when applying this scheme to a semi-discrete system with periodic boundary conditions.
- Demonstrate that the second-order accurate upwind method is stable for CFL numbers less than 1 when used with the explicit Euler scheme or the two-stage Runge-Kutta scheme.
- Describe one significant problem with using the second-order accurate upwind flux approximation and the two-stage Runge-Kutta time advance scheme.
- Explain how to apply an upstream boundary condition for the wave equation and why a downstream boundary condition is not needed.
- Explain how to decide at what time to enforce a time-dependent boundary condition.

7.1 Boundary Conditions for the Wave Equation

We have discussed in detail how to compute fluxes for the wave equation in the interior of a computational domain. What should one do at the boundaries? Let us consider first the analytic problem for a finite domain, which is properly posed as:

$$\begin{aligned} \frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} &= 0 \\ 0 \le x \le L & \qquad 0 \le t \\ T(x,0) &= f(x) \\ T(0,t) &= g(t) \end{aligned}$$

The exact solution to this problem is

$$T(x,t) = \begin{cases} f(x-ut) & x > ut \\ g\left(t - \frac{x}{u}\right) & x < ut \end{cases}$$

Three key observations are appropriate here.

1. The solution propagates strictly from left to right, which implies that fluxes should be calculated using data from the left ("upwind") so that the numerical solution will behave in the same way as the mathematical solution. This also explains the lack of a boundary condition at the right boundary. For strictly upwind schemes, we can evaluate the flux at $i_{\max+\frac{1}{2}}$ just as the normal interior fluxes for use in the flux integral for CV $i_{\max}$.

2. The flux at $\frac{3}{2}$ can not always be evaluated using the interior flux scheme (notably for second-order upwind schemes).

3. The boundary condition at $x = 0$ is sufficient for us to compute the flux there, although this flux varies in time.

7.1.1 Flux evaluation at $\frac{3}{2}$ (first interior interface)

For the second-order upwind scheme and its close variants, this flux would be evaluated by using extrapolation to estimate

$$T_{\frac{3}{2}} \approx \frac{3\bar{T}_1 - \bar{T}_0}{2}$$

Alas, we do not have a control volume 0 to use in this context. We could choose to use a first-order accurate flux here (e.g., $T_{\frac{3}{2}} \approx \bar{T}_1$), or we could choose to use a centered flux evaluation $\left(\text{i.e., } T_{\frac{3}{2}} = \frac{\bar{T}_2+\bar{T}_1}{2}\right)$. We could also extrapolate the temperature using the boundary condition evaluated at an appropriate time:

$$T_{\frac{3}{2}} \approx 2\bar{T}_1 - g(t)$$

Equivalently, and possibly easier to code, we could use a ghost cell and set

$$\bar{T}_0 = 2g(t) - \bar{T}_1$$

7.1.2 Flux evaluation at $x = 0$

At $x = 0$, the flux is known from the boundary condition, because $T(0,t) = g(t)$. In practice, this means that we have to be careful to evaluate the flux at the correct time, which is always the same time as the flux integral is evaluated. For example,

for the first-order explicit Euler time advance scheme, the flux integral is evaluated at time level n, so we use $T^n_{\frac{1}{2}} = T(x = 0, n\Delta t) = g(n\Delta t)$.

At what time should the boundary flux be evaluated for each stage of the two-stage Runge-Kutta scheme of Section 6.4.5? What about the four-stage scheme of Problem 6.5?

7.2 Basic Results for the Wave Equation

Consider a simple test case for the wave equation: propagation of a sine wave at unit speed in the domain $[0, 1]$ on a mesh of forty control volumes with periodic boundary conditions until time $t = 1$. At this time, the wave should be back to precisely its starting position.

Suppose that we use explicit Euler time advance and two space discretization schemes: first-order upwind and second-order centered. In each case, the CFL number is 0.75. Figure 7.1 shows the results for this test. For this time advance method, the centered space discretization is unstable. Almost as bad is the first-order upwind scheme, which damps out the solution rather quickly.

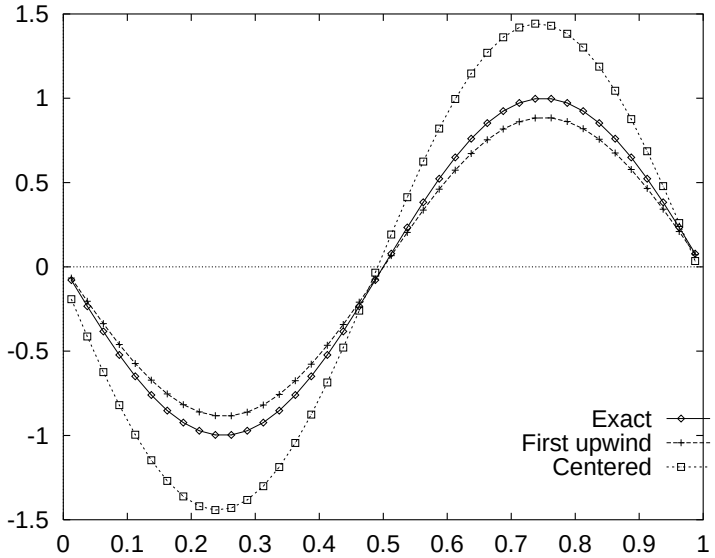


Figure 7.1: First-order time advance for the wave equation with several space discretizations

Suppose that we replace the explicit Euler time discretization with the second-order Runge-Kutta time advance scheme, leaving everything else the same. The result of propagating a sine wave with this time advance scheme is shown in Figure 7.2. The first-order upwind discretization gives even poorer results here than for the previous case. The second-order accurate schemes both do quite well, although the centered scheme is still ever so slightly unstable. The reason that this instability is not yet visible is that the amplification factor is very near one. Note also that the second-order upwind scheme has what is known as a leading phase error for these wave length: the wave propagates a bit faster than it should.

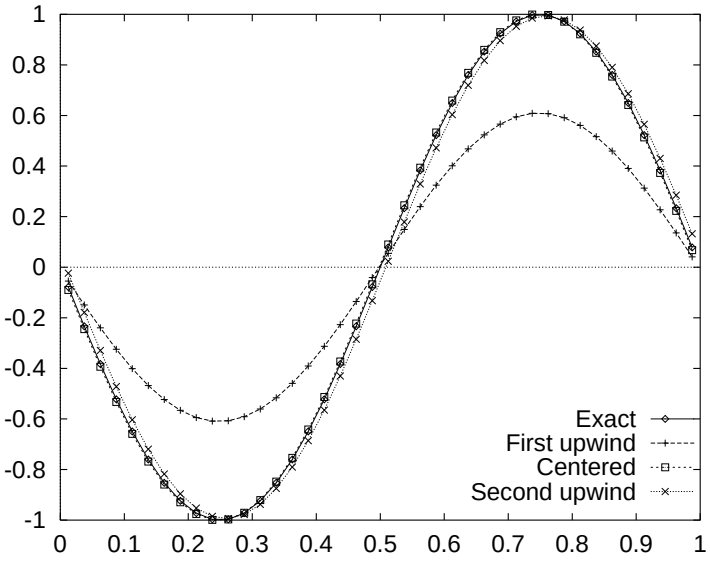


Figure 7.2: Second-order time advance for the wave equation with several space discretizations

Now that we have a scheme[1] that works well for this case for the wave equation, are we done?

No.

Suppose we were to propagate a square wave instead of a sine wave (see Figure 7.3). The centered difference scheme is clearly unstable, and the first-order upwind scheme is clearly damping the solution very rapidly. The second-order upwind scheme is not *too* bad in comparison, except for the presence of significant overshoots.

[1] Or two, if our simulations are short enough that the unstable centered scheme is okay.

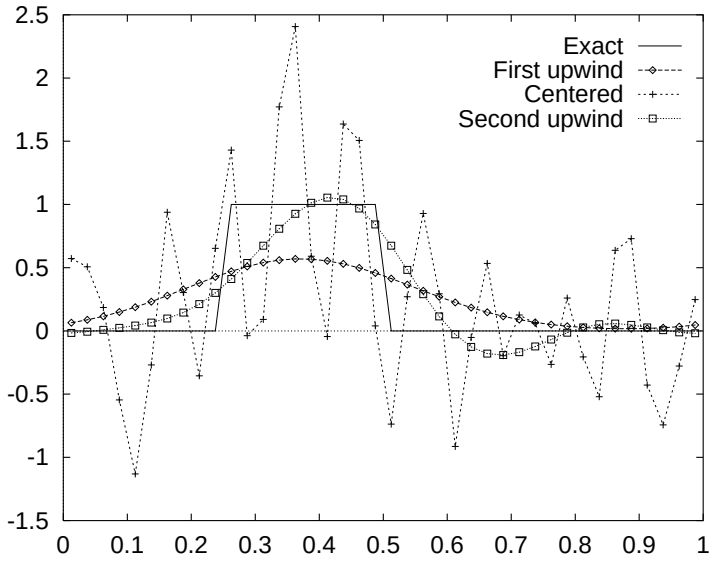


Figure 7.3: Second-order time advance for the wave equation propagating a square wave.

If we use a much finer mesh with the second-order upwind scheme, we don't get results that look any better, as shown in Figure 7.4.

Clearly, something needs to be done about this problem, but what?

7.3 Advanced Schemes for the Wave Equation

There are several families of schemes that can be used to treat the overshoots we just saw; these schemes differ both in the philosophy of the approach used and in their success for retaining accuracy for smooth solutions.

Learning Objectives. Students will be able to:

- Define monotonicity in the context of discretization schemes for partial differential equations.
- List four approaches for obtaining monotone solutions for the wave equation that are nominally second-order accurate.
- Describe how limited extrapolation computes cell interface data to ensure a monotone solution.

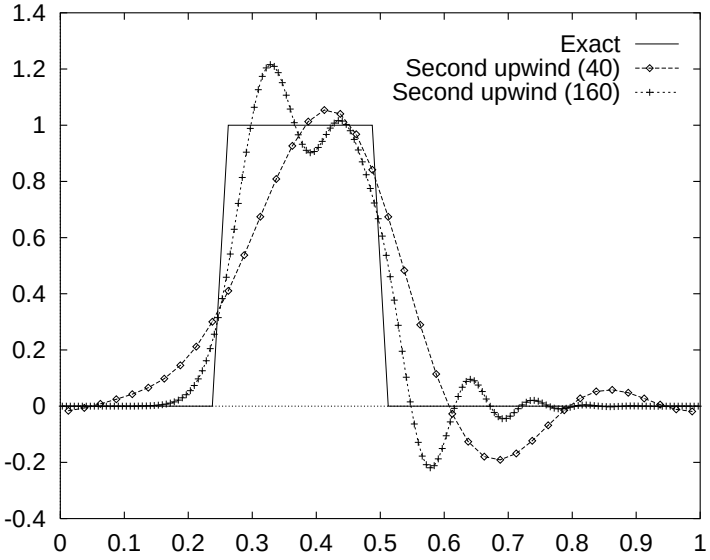


Figure 7.4: Effect of mesh refinement on square wave propagation using the second-order upwind scheme.

- Describe in general terms how total-variation diminishing schemes adjust the anti-diffusive flux added to a first-order upwind discretization to maintain monotonicity.

- Describe the principle that essentially non-oscillatory schemes are based on and explain the advantages of this type of flux calculation.

- Describe how and why flux-corrected transport schemes apply anti-diffusion to reduce diffusion while maintaining monotonicity.

7.3.1 Limited Extrapolation

Limited extrapolation seeks to eliminate overshoots and undershoots in the solution of the wave equation by not allowing new local extrema to arise in extrapolating to compute the flux at $i+\frac{1}{2}$. That is, if

$$T_{i+\frac{1}{2}} = \frac{3\overline{T}_i - \overline{T}_{i-1}}{2} > \max\left(\overline{T}_i, \overline{T}_{i+1}\right)$$

then $T_{i+\frac{1}{2}}$ is replaced by $\max\left(\overline{T}_i, \overline{T}_{i+1}\right)$. Likewise, if

$$T_{i+\frac{1}{2}} = \frac{3\overline{T}_i - \overline{T}_{i-1}}{2} < \min\left(\overline{T}_i, \overline{T}_{i+1}\right)$$

then $T_{i+\frac{1}{2}}$ is replaced by $\min\left(\overline{T}_i, \overline{T}_{i+1}\right)$. Another way of putting this mathematically is

$$T_{i+\frac{1}{2}} = \begin{cases} \max\left(\overline{T}_i, \overline{T}_{i+1}\right) & \text{if } \frac{3\overline{T}_i - \overline{T}_{i-1}}{2} > \max\left(\overline{T}_i, \overline{T}_{i+1}\right) \\ \min\left(\overline{T}_i, \overline{T}_{i+1}\right) & \text{if } \frac{3\overline{T}_i - \overline{T}_{i-1}}{2} < \min\left(\overline{T}_i, \overline{T}_{i+1}\right) \\ \frac{3\overline{T}_i - \overline{T}_{i-1}}{2} & \text{otherwise} \end{cases}$$

Yet another choice is to write this as a single series of max's and min's:

$$T_{i+\frac{1}{2}} = \max\left(\min\left(\overline{T}_i, \overline{T}_{i+1}\right), \min\left(\frac{3\overline{T}_i - \overline{T}_{i-1}}{2}, \max\left(\overline{T}_i, \overline{T}_{i+1}\right)\right)\right)$$

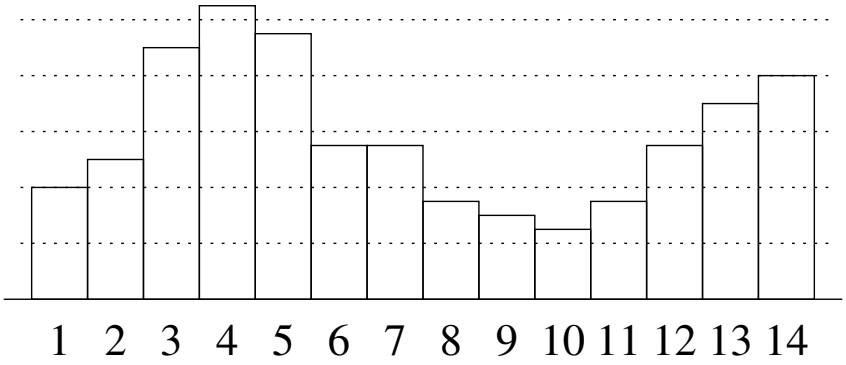


Figure 7.5: Example control-volume averaged solution

In Figure 7.5, the limiter will clearly be active at $4\frac{1}{2}$ and $10\frac{1}{2}$, to prevent the increase of a local maximum and decrease of a local minimum, respectively. The value used at $3\frac{1}{2}$ will be limited to the value in CV 4 for the same reason. The value used at $6\frac{1}{2}$ will be limited to the common value in control volumes 6 and 7. Finally — and most subtly — the value at $8\frac{1}{2}$ without limiting would be lower than the CV average value in CV 9, although higher than in CV 10; this value would be limited to exactly the average in CV 9.

7.3.2 Total-Variation Diminishing (TVD) Schemes

The total variation of a solution in one dimension is defined as the sum of the absolute values of the change in solution between successive extrema. Referring again to Figure 7.5, the total variation in this solution would be

$$TV = (\bar{T}_4 - \bar{T}_1) + (\bar{T}_4 - \bar{T}_{10}) + (\bar{T}_{14} - \bar{T}_{10}) \tag{7.1}$$

Total-variation diminishing (TVD) schemes have the property that, while individual extrema may get higher or lower, the total variation is non-increasing. That is, the value of $\bar{T}_{10}$ might drop, but the total variation of Equation 7.1 would not increase. This implies that either $\bar{T}_1$ must increase or $\bar{T}_4$ or $\bar{T}_{14}$ must decrease.

Upwind TVD schemes use a flux that (for the wave equation) can be written as

$$F_{i+\frac{1}{2}} = u\left[\bar{T}_i + \frac{\psi_{i+\frac{1}{2}}}{2}(\bar{T}_i - \bar{T}_{i-1})\right] \tag{7.2}$$

Note that for $\psi_{i+\frac{1}{2}} = 0$ this reduces to the first-order upwind flux, while for $\psi_{i+\frac{1}{2}} = 1$ we get the second-order upwind flux. The key element of upwind TVD schemes is the definition of ψ; this determines how much anti-diffusion to use (and therefore how much diffusion remains). $\psi_{i+\frac{1}{2}}$ is written as a function of

$$r_{i+\frac{1}{2}} \equiv \frac{\bar{T}_{i+1} - \bar{T}_i}{\bar{T}_i - \bar{T}_{i-1}}$$

It can be shown that the acceptable range for ψ is the region outlined in Figure 7.6. Values of r less than zero indicate that CV i is a local extremum; here ψ must be zero to avoid accentuating the extremum.

If r lies between 0 and 1, the magnitude of the slope is decreasing from $i-1$ to i to $i+1$, as for $i = 3$ in Figure 7.5. For the maximum allowable value of ψ, $2r$, Equation 7.2 reduces to

$$\begin{aligned} F_{i+\frac{1}{2}} &= u\left[\bar{T}_i + \frac{2r_{i+\frac{1}{2}}}{2}(\bar{T}_i - \bar{T}_{i-1})\right] \\ &= u\left[\bar{T}_i + \frac{\bar{T}_{i+1} - \bar{T}_i}{\bar{T}_i - \bar{T}_{i-1}}(\bar{T}_i - \bar{T}_{i-1})\right] \\ &= u\bar{T}_{i+1} \end{aligned}$$

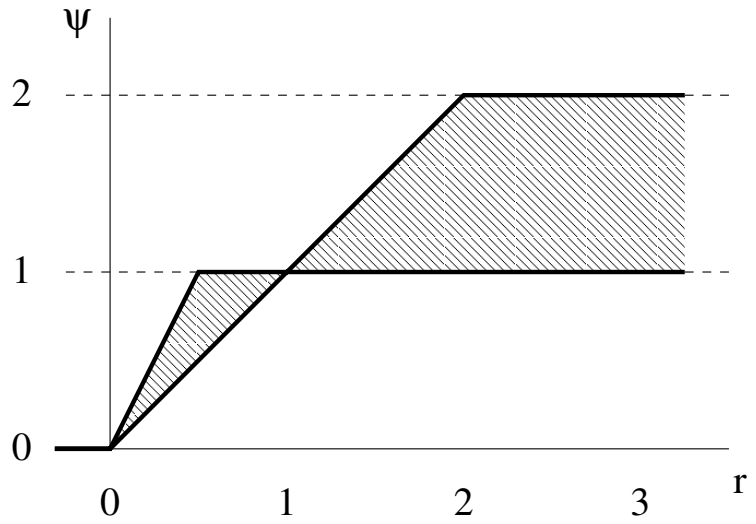


Figure 7.6: Legal values of $\psi(r)$ for TVD schemes

This applies only in cases where the simple extrapolation ($\psi = 1$) creates a new extremum $\left(r \leq \frac{1}{2}\right)$. For larger values of r, the extrapolation can be used without limiting. The lower boundary of the TVD region for decreasing slope, $\psi = r$, corresponds to using a central flux:

$$F_{i+\frac{1}{2}} = u\frac{\bar{T}_i + \bar{T}_{i+1}}{2}$$

If $r \geq 1$, the magnitude of the slope is increasing from $i-1$ to i to $i+1$, as for $i = 11$ in Figure 7.5. For such control volumes, the second-order upwind scheme can be used directly ($\psi = 1$). For moderate values of r, ($1 \leq r \leq 2$), the centered scheme ($\psi = r$) satisfies the TVD requirements, but discretizations with largely downwind dependence do not. For large r (≥ 2), ψ is required to remain below 2. For $\psi = 2$, the TVD flux becomes

$$\begin{aligned} F_{i+\frac{1}{2}} &= u\left[\bar{T}_i + \frac{2}{2}(\bar{T}_i - \bar{T}_{i-1})\right] \\ &= u\left[2\bar{T}_i - \bar{T}_{i-1}\right] \end{aligned}$$

This is the value one would obtain by extrapolating from CV's i and $i-1$ to the center of CV $i+1$; consequently, this flux choice is more aggressive in steepening smooth gradients than lower values of ψ.

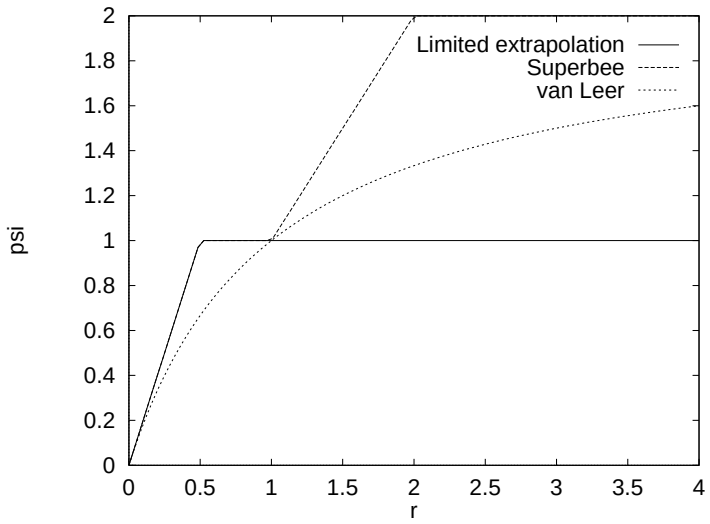


Figure 7.7: Three TVD limiters

Figure 7.7 shows ψ as a function of r for three cases. The limited extrapolation scheme can be written as a TVD scheme with

$$\psi(r) = \begin{cases} 0 & \text{if } r \leq 0 \\ 2r & \text{if } r \leq \frac{1}{2} \\ 1 & \text{otherwise} \end{cases} \tag{7.3}$$

The Superbee scheme follows the top of the allowable range for ψ, making it the most "compressive", or slope-steepening, of all possible TVD schemes. ψ for this case can be written as

$$\psi(r) = \begin{cases} 0 & \text{if } r \leq 0 \\ 2r & \text{if } r \leq \frac{1}{2} \\ 1 & \text{if } r \leq 1 \\ r & \text{if } r \leq 2 \\ 2 & \text{otherwise} \end{cases} \tag{7.4}$$

Finally, van Leer's scheme is a smoothly varying scheme with asymptotic behavior for large r that matches Superbee:

$$\psi(r) = \frac{r + |r|}{1 + r} \tag{7.5}$$

TVD schemes can be extended to higher dimensions by applying the flux calculation direction-by-direction. That is, in two dimensions, the flux at $i + \frac{1}{2}, j$ is calcu-

lated by using data from $i-1, j$, i, j, and $i+1, j$. Likewise, the flux at $i, j+\frac{1}{2}$ is calculated by using data from $i, j-1$, i, j, and $i, j+1$.

7.3.3 Flux-corrected Transport (FCT) Schemes

Flux-corrected transport schemes deliberately introduce enough numerical dissipation to produce monotone solutions, then cancel as much of that dissipation as possible without producing overshoots. The prototypical flux-corrected transport (FCT) scheme — called SHASTA — was designed as a second-order accurate method that had been deliberately "broken" by the addition of extra dissipation. The flux for the wave equation using this scheme is

$$F_{i+\frac{1}{2}} = u\left[\frac{\bar{T}_i + \bar{T}_{i+1}}{2} - \left(\frac{1}{8} + \frac{u\Delta t}{2\Delta x}\right)(\bar{T}_{i+1} - \bar{T}_i)\right] \tag{7.6}$$

Without the velocity-independent dissipative flux, this is precisely the *Lax-Wendroff* scheme, which can be shown to be second-order accurate in time and space.[2] The flux of Equation 7.6 is used to produce an interim solution at time level $n+1$:

$$\tilde{T}_i^{n+1} = \tilde{T}_i^n - \frac{u\Delta t}{\Delta x}\left(F^n_{i+\frac{1}{2}} - F^n_{i-\frac{1}{2}}\right) \tag{7.7}$$

The first differences of the interim solution are computed:

$$\Delta_{i+\frac{1}{2}} = \tilde{T}^{n+1}_{i+1} - \tilde{T}^{n+1}_i$$

and used to produce an anti-diffusive flux:

$$F^{ad}_{i+\frac{1}{2}} = S \cdot \max\left(0, \min\left(S\Delta_{i-\frac{1}{2}}, \frac{1}{8}\left|\Delta_{i+\frac{1}{2}}\right|, S\Delta_{i+\frac{3}{2}}\right)\right)$$

where $S \equiv \text{sign}\left(\Delta_{i+\frac{1}{2}}\right)$.

Note that the anti-diffusive flux has some similarities in form to the TVD diffusive fluxes, in that in both cases, existing extrema are protected by using a first-order accurate monotone scheme at extrema. For smooth solutions, the anti-diffusive flux

[2]This proof uses different methods than we have discussed in class, because the Lax-Wendroff scheme can not, strictly speaking, be derived using the semi-discrete formulation: it uses two-stage time advance with different space discretization for the two stages.

is carefully designed to precisely cancel all of the diffusion added in the first step, giving a scheme that is second-order accurate.

The solution at time level $n+1$ is then computed using

$$\bar{T}_i^{n+1} = \tilde{T}_i^{n+1} - \left(F^{ad}_{i+\frac{1}{2}} - F^{ad}_{i-\frac{1}{2}} \right) \tag{7.8}$$

Despite having excellent properties in one dimension, flux-corrected transport schemes are not popular in two or three dimensions, because the geometric interpretations of advanced FCT anti-diffusive flux formulae do not generalize well.

7.3.4 Essentially Non-Oscillatory (ENO) Schemes

Simple extrapolation schemes for flux calculation for the wave equation, blindly applied, lead to overshoots and undershoots because extrapolation implicitly assumes that there is a smooth underlying function to extrapolate — that the solution can be expanded locally in a Taylor series. This is not true near discontinuities, and the three previous families of schemes are all ways to fix the extrapolation so that it is at least moderately well-behaved near discontinuities.

Essentially non-oscillatory (ENO) schemes take a different approach. Instead of trying to fix a mathematically invalid Taylor series expansion, ENO schemes seek to construct a valid extrapolation using smooth data. More precisely, an ENO scheme of order k produces an extrapolation that is k-th order accurate for smooth solutions and has overshoots that are no larger than $O\left(\Delta x^{k-1}\right)$.

There are a number of variations in the details of how to accomplish this. I will outline a technique I developed for unstructured, multi-dimensional meshes. The simplification to one-dimensional, equally-space meshes is more complex than some other schemes, but not outrageously so.

Suppose that there are no discontinuities in the solution. Then computing a flux using

$$F_{i+\frac{1}{2}} = u\left(\bar{T}_i + \frac{\bar{T}_{i+1} - \bar{T}_{i-1}}{4} \right) \tag{7.9}$$

can be shown to be second-order accurate. Unfortunately, this approach allows large overshoots near discontinuities. For example, in Figure 7.5, the extrapolated value at $6\frac{1}{2}$ is an overshoot. Because the jump from CV 5 to CV 6 is of $O(1)$, so is

the overshoot. Suppose we rewrite Equation 7.9 as

$$F_{i+\frac{1}{2}} = u\left[\bar{T}_i + \frac{\Delta x}{2}\left(a\frac{\bar{T}_{i+1}-\bar{T}_i}{\Delta x} + (1-a)\frac{\bar{T}_i-\bar{T}_{i-1}}{\Delta x}\right)\right] \tag{7.10}$$

If $a = \frac{1}{2}$, Equations 7.9 and 7.10 are identical. This flux is second-order accurate for all values of a, which gives us the freedom to choose a in order to prevent overshoots from being too large.

Consider first the case of smooth extrema — where the numerical second derivative

$$\lim_{\Delta x\to 0}\frac{\Delta^2 T}{\Delta x^2} \equiv \frac{\bar{T}_{i+1}-2\bar{T}_i+\bar{T}_{i-1}}{\Delta x^2} \to \frac{d^2T}{dx^2}$$

is bounded and the one-sided differences

$$\begin{aligned}\frac{\Delta_+ T}{\Delta x}_i &\equiv \frac{\bar{T}_{i+1}-\bar{T}_i}{\Delta x}\\ \frac{\Delta_- T}{\Delta x}_i &\equiv \frac{\bar{T}_i-\bar{T}_{i-1}}{\Delta x}\end{aligned}$$

are also bounded. In this case, even though small overshoots may be present, the size of the overshoots can be shown to be $O\left(\Delta x^2\right)$, which is acceptable for an ENO scheme.

If there is a discontinuity of $O(1)$ between i and (say) $i-1$, on the other hand, the numerical second derivative is not bounded:

$$\lim_{\Delta x\to 0}\frac{\Delta^2 T}{\Delta x^2} \equiv \frac{\bar{T}_{i+1}-2\bar{T}_i+\bar{T}_{i-1}}{\Delta x^2} \sim \frac{1}{\Delta x^2}$$

One first derivative is bounded while the other is not:

$$\begin{aligned}\frac{\Delta_+ T}{\Delta x}_i &\equiv \frac{\bar{T}_{i+1}-\bar{T}_i}{\Delta x} = O(1)\\ \frac{\Delta_- T}{\Delta x}_i &\equiv \frac{\bar{T}_i-\bar{T}_{i-1}}{\Delta x} \sim \frac{1}{\Delta x}\end{aligned}$$

We would prefer to ignore the data from the left-hand control volume, as it is clearly (from a human viewpoint) irrelevant. To do this computationally, we must choose a carefully. One simple function that works well is

$$w_+ = \frac{1}{1+C\left|\frac{\Delta^2 T}{\Delta x^2}\right|\left(\frac{\Delta_+ T}{\Delta x}_i\right)^2\Delta x^2}$$

$$
\begin{aligned}
w_- &= \frac{1}{1 + C\left|\frac{\Delta^2 T}{\Delta x^2}\right| \left(\frac{\Delta_- T}{\Delta x}_i\right)^2 \Delta x^2} \\
a &= \frac{w_-}{w_+ + w_-}
\end{aligned}
$$

It is easy to show that this choice of a gives

- Equation 7.9 in smooth regions, where all finite differences are bounded with mesh refinement; more precisely, $a \to \frac{1}{2} + O\left(\Delta x^2\right)$.
- $a \to 1 - O\left(\Delta x^2\right)$ when a discontinuity exists between control volumes i and $i-1$; this is a one-sided extrapolation with (more difficult but still provable) small overshoots.
- $a \to O\left(\Delta x^2\right)$ when a discontinuity exists between control volumes i and $i+1$; this is a one-sided extrapolation from the other side.

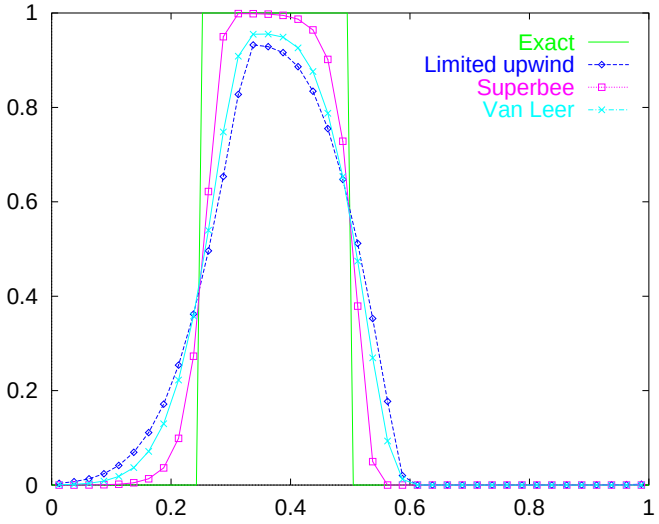
Note that a particularly clever choice of a would give you a third-order accurate flux at the interface for smooth solutions. I personally don't use this choice. My interest in this problem comes from the area of reconstruction on multidimensional unstructured meshes, where the extension of this order increase is difficult or impossible, so I don't bother even in one dimension.

7.3.5 Sample Calculations

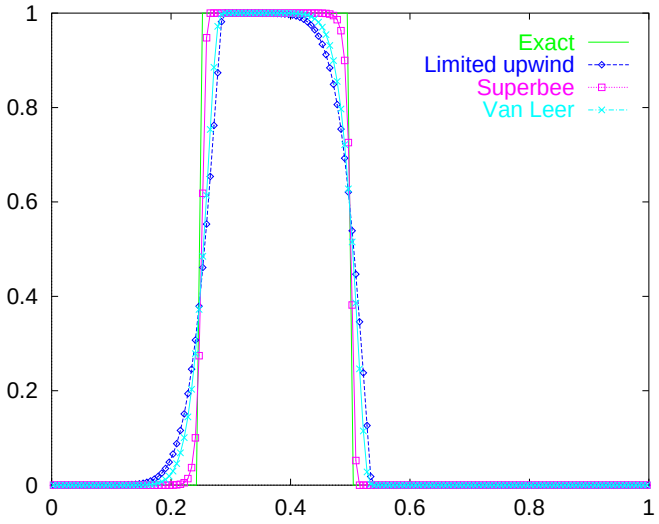
Figures 7.8–7.10 show the results of applying all these schemes to the wave equation with both square wave and sine wave initial data. In all cases, the boundary conditions are periodic and the solution is advanced in time until the wave has traveled around the mesh exactly once.

For the square wave, either the Superbee TVD scheme or the SHASTA FCT scheme gives the best results, with the others also performing fairly well. For the sine wave problem, the ENO and FCT schemes give the best results because of their superior performance near smooth extrema.

Which scheme is best overall? That's a very hard call, especially on the basis of only two test problems. Each approach has its strengths and weaknesses. ***Danger: What follows is opinion and should not be construed as a consensus among researchers or practitioners in CFD.*** My personal favorite is the TVD schemes. TVD schemes generalize to arbitrary unstructured meshes much better than FCT schemes, while also giving much better steady-state convergence than ENO schemes.

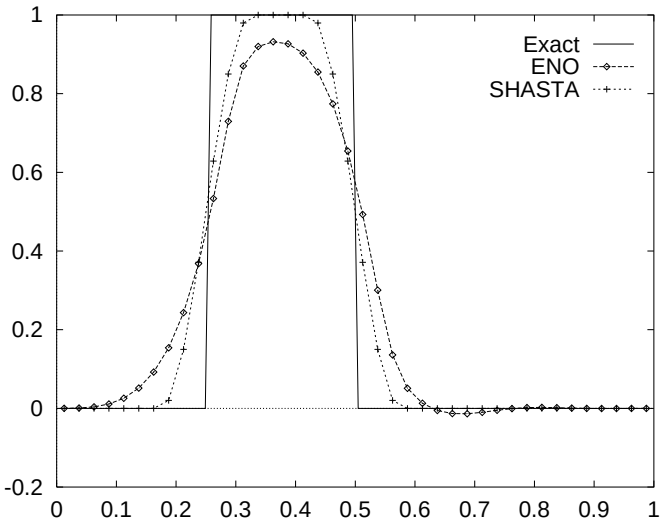


(a) Coarse mesh

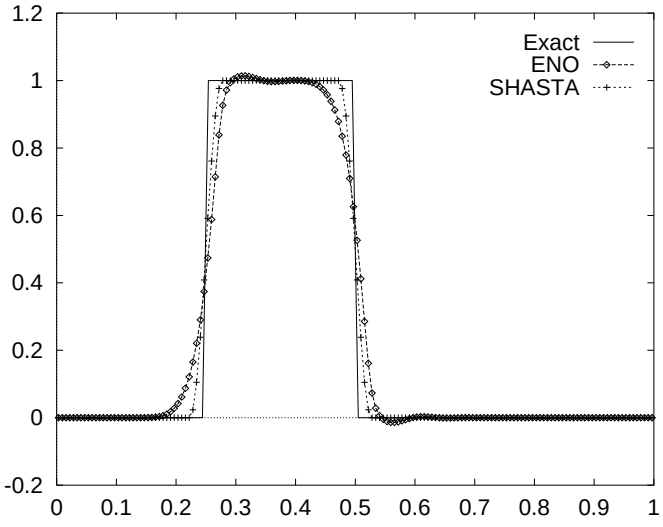


(b) Fine mesh

Figure 7.8: Upwind TVD schemes (square wave)

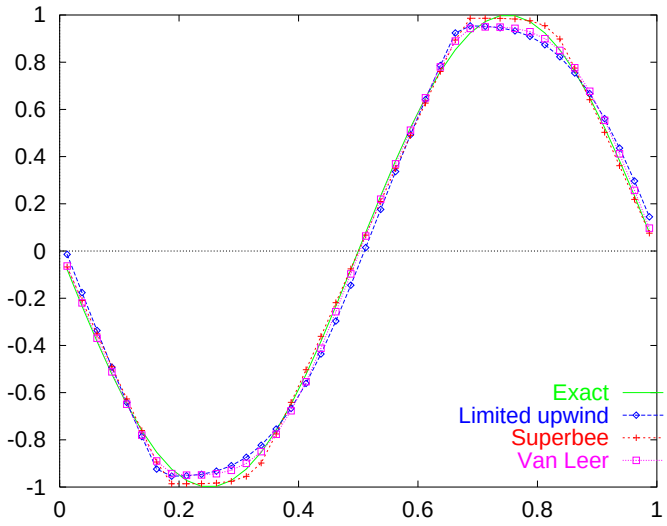


(a) Coarse mesh

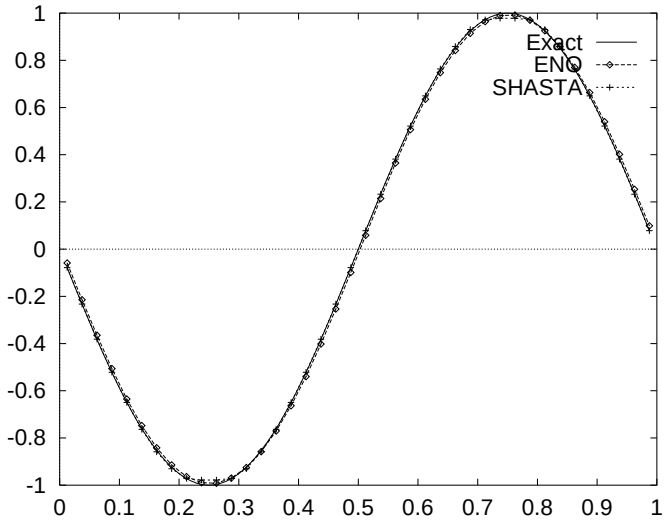


(b) Fine mesh

Figure 7.9: ENO and FCT schemes (square wave)



(a) TVD schemes



(b) ENO and FCT schemes

Figure 7.10: Propagation of a smooth solution (sine wave)

Chapter 8

The Incompressible Energy Equation

The incompressible energy equation is useful both in its own right for predicting energy transfer and to provide experience combining convective and diffusive terms in the same governing equation before moving on to the Navier-Stokes equations, which are mathematically similar in many ways but have the added complication of being a coupled system of equations.

Also, the same equation can be used to model other physical processes. For example, if T is interpreted as a chemical species concentration and the viscous dissipation term on the right-hand side of the energy equation (see below) is replaced by an appropriate source term, then the energy equation correctly model species concentration in chemically reacting flow.

The differential form of the incompressible energy equation can be written as:

$$\begin{aligned}\frac{\partial T}{\partial t}+\frac{\partial uT}{\partial x}+\frac{\partial vT}{\partial y} &= \frac{1}{Re\cdot Pr}\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right)\\ &\quad +\frac{Ec}{Re}\left(2\left(\frac{\partial u}{\partial x}\right)^2+2\left(\frac{\partial v}{\partial y}\right)^2+\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right)^2\right)\end{aligned}$$

Applying Gauss's Theorem over an arbitrary fixed control volume, we can arrive at the integral form of the energy equation:

$$\frac{\partial \bar{T}}{\partial t}A+\oint_{\partial CV}\vec{v}T\cdot\vec{n}ds = \frac{1}{Re\cdot Pr}\oint_{\partial CV}\nabla T\cdot\vec{n}ds$$

$$+\frac{Ec}{Re}\int_{CV}\left(2\left(\frac{\partial u}{\partial x}\right)^2+2\left(\frac{\partial v}{\partial y}\right)^2+\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right)^2\right)dA$$

For a finite volume in a 2D uniform mesh:

$$\begin{aligned}
\frac{d\bar{T}_{i,j}}{dt}\Delta x\Delta y \qquad & \frac{1}{Re\cdot Pr}\left(\frac{\partial T}{\partial x}^{i+\frac{1}{2},j}_{i-\frac{1}{2},j}\Delta y+\frac{\partial T}{\partial y}^{i,j+\frac{1}{2}}_{i,j-\frac{1}{2}}\Delta x\right)\\
+(uT)^{i+\frac{1}{2},j}_{i-\frac{1}{2},j}\Delta y \;=\; & +\frac{Ec}{Re}\left(2\left(\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x}\right)^2+2\left(\frac{v_{i,j+1}-v_{i,j-1}}{2\Delta y}\right)^2\right)\Delta x\Delta y\\
+(vT)^{i,j+\frac{1}{2}}_{i,j-\frac{1}{2}}\Delta x \qquad & +\frac{Ec}{Re}\left(\frac{v_{i+1,j}-v_{i-1,j}}{2\Delta x}+\frac{u_{i,j+1}-u_{i,j-1}}{2\Delta y}\right)^2\Delta x\Delta y
\end{aligned}$$

Abbreviating the (constant) viscous dissipation terms as $S_{i,j}$ and combining the convective and diffusive fluxes, we arrive at last at a fairly compact form of the equation.

$$\frac{d\bar{T}_{i,j}}{dt}+\frac{1}{\Delta x}\left(uT-\frac{1}{Re\cdot Pr}\frac{\partial T}{\partial x}\right)^{i+\frac{1}{2},j}_{i-\frac{1}{2},j}+\frac{1}{\Delta y}\left(vT-\frac{1}{Re\cdot Pr}\frac{\partial T}{\partial y}\right)^{i,j+\frac{1}{2}}_{i,j-\frac{1}{2}}=S_{i,j} \quad (8.1)$$

8.1 Simple Discretization of the Incompressible Energy Equation

8.2 Time Discretization of the Energy Equation

8.2.1 Implicit Euler time advance applied to the energy equation

If we write the energy equation in fully-discrete form using the implicit Euler time advance scheme, we arrive at the following equation:

$$\begin{aligned}\frac{\bar{T}_{i,j}^{n+1}-\bar{T}_{i,j}^{n}}{\Delta t} &= -\frac{1}{\Delta x}\left(\frac{u_{i+1,j}\bar{T}_{i+1,j}^{n+1}-u_{i-1,j}\bar{T}_{i-1,j}^{n+1}}{2}\right.\\ &\quad\left.-\frac{1}{Re\cdot Pr}\left(\frac{\bar{T}_{i+1,j}^{n+1}-2\bar{T}_{i,j}^{n+1}+\bar{T}_{i-1,j}^{n+1}}{\Delta x}\right)\right)\\ &\quad-\frac{1}{\Delta y}\left(\frac{v_{i,j+1}\bar{T}_{i,j+1}^{n+1}-v_{i,j-1}\bar{T}_{i,j-1}^{n+1}}{2}\right.\\ &\quad\left.-\frac{1}{Re\cdot Pr}\left(\frac{\bar{T}_{i,j+1}^{n+1}-2\bar{T}_{i,j}^{n+1}+\bar{T}_{i,j-1}^{n+1}}{\Delta y}\right)\right)\\ &\quad+S_{i,j}\end{aligned}$$

In practice, we will typically want to write this in what is called δ-form by replacing $T_{i,j}^{n+1}\equiv T_{i,j}^{n}+\delta T_{i,j}^{n+1}$ and simplifying. δ-form is much more convenient near steady state, where round off errors in the calculation of T can easily exceed δT, the change in T from one time level to the next. Also, as we'll see later in the course, δ-form is more convenient for non-linear problems.

$$\begin{aligned}\frac{\overline{\delta T}_{i,j}}{\Delta t} &+ \frac{1}{\Delta x}\left(\frac{u_{i+1,j}\overline{\delta T}_{i+1,j}-u_{i-1,j}\overline{\delta T}_{i-1,j}}{2}\right.\\ &\quad\left.-\frac{1}{Re\cdot Pr}\left(\frac{\overline{\delta T}_{i+1,j}-2\overline{\delta T}_{i,j}+\overline{\delta T}_{i-1,j}}{\Delta x}\right)\right)\\ &+ \frac{1}{\Delta y}\left(\frac{v_{i,j+1}\overline{\delta T}_{i,j+1}-v_{i,j-1}\overline{\delta T}_{i,j-1}}{2}\right.\end{aligned}$$

$$
\begin{aligned}
& -\frac{1}{Re\cdot Pr}\left(\frac{\overline{\delta T}_{i,j+1}-2\overline{\delta T}_{i,j}+\overline{\delta T}_{i,j-1}}{\Delta y}\right)\Bigg) \\
= & -\frac{1}{\Delta x}\left(\frac{u_{i+1,j}\bar{T}^n_{i+1,j}-u_{i-1,j}\bar{T}^n_{i-1,j}}{2}\right. \\
& \left.-\frac{1}{Re\cdot Pr}\left(\frac{\bar{T}^n_{i+1,j}-2\bar{T}^n_{i,j}+\bar{T}^n_{i-1,j}}{\Delta x}\right)\right) \\
- & \frac{1}{\Delta y}\left(\frac{v_{i,j+1}\bar{T}^n_{i,j+1}-v_{i,j-1}\bar{T}^n_{i,j-1}}{2}\right. \\
& \left.-\frac{1}{Re\cdot Pr}\left(\frac{\bar{T}^n_{i,j+1}-2\bar{T}^n_{i,j}+\bar{T}^n_{i,j-1}}{\Delta y}\right)\right) \\
+ & S_{i,j}
\end{aligned}
\tag{8.2}
$$

The right-hand side is the flux integral evaluated at time level n.

8.2.2 Trapezoidal time advance applied to the energy equation

8.3 Boundary Conditions

Boundary conditions for the energy equation fall into two categories: those that are analogous to Poisson equation boundary conditions and those that aren't. Generally speaking, we can use the same approach for temperature boundary conditions at walls as for the Poisson equation (see Section 4.2). For inflow and outflow boundaries, there are differences.

8.3.1 Inflow boundaries

At inflow boundaries, the temperature is usually specified in terms of known upstream values. Because the temperature at the boundary is know, the flux across the boundary can easily be calculated. Generally this is most easily done by setting a ghost cell value in the same way as for Dirichlet boundary conditions for the Poisson equation(see Section 4.2.2).

8.3.2 Outflow boundaries

At outflow boundaries, there are two fairly common choices for boundary conditions. First, the temperature at the outflow may be determined strictly by the upstream temperature in the computational domain as fluid convects out; in this case, the temperature for the ghost cell would typically be extrapolated from interior data:

$$T_{i_{\max}+1,j} = 2T_{i_{\max},j} - T_{i_{\max}-1,j}$$

The other common choice is to assume that the temperature gradient is zero normal to the boundary; this is completely equivalent to a Neumann boundary condition applied to the Poisson equation (see Section 4.2.1).

8.4 Approximate Factorization

Both implicit discretizations for the energy equation require the solution of a matrix that does not have a simple, tightly-banded structure. However, it is possible to factor this matrix approximately into two scalar tri-diagonal matrices using a technique called *approximate factorization.*

Learning Objectives. Students will be able to:

- Describe the approximate factorization scheme as applied to the implicit Euler and trapezoidal time advance schemes. Prove that approximate factorization does not affect the order of time accuracy.
- Outline how to advance the energy equation in time using an implicit time advance scheme and ADI.

Regardless of how we choose to do the space discretization for the two-dimensional energy equation, we will end up with a semi-discrete equation of the form:

$$\frac{d}{dt}\begin{pmatrix} \bar{T}_{1,1} \\ \bar{T}_{2,1} \\ \bar{T}_{3,1} \\ \vdots \\ \bar{T}_{i-1,j} \\ \bar{T}_{i,j} \\ \bar{T}_{i+1,j} \\ \vdots \\ \bar{T}_{i_{\max},j_{\max}} \end{pmatrix} + [D_x]\begin{pmatrix} \bar{T}_{1,1} \\ \bar{T}_{2,1} \\ \bar{T}_{3,1} \\ \vdots \\ \bar{T}_{i-1,j} \\ \bar{T}_{i,j} \\ \bar{T}_{i+1,j} \\ \vdots \\ \bar{T}_{i_{\max},j_{\max}} \end{pmatrix} + [D_y]\begin{pmatrix} \bar{T}_{1,1} \\ \bar{T}_{2,1} \\ \bar{T}_{3,1} \\ \vdots \\ \bar{T}_{i-1,j} \\ \bar{T}_{i,j} \\ \bar{T}_{i+1,j} \\ \vdots \\ \bar{T}_{i_{\max},j_{\max}} \end{pmatrix} = \begin{pmatrix} \bar{S}_{1,1} \\ \bar{S}_{2,1} \\ \bar{S}_{3,1} \\ \vdots \\ \bar{S}_{i-1,j} \\ \bar{S}_{i,j} \\ \bar{S}_{i+1,j} \\ \vdots \\ \bar{S}_{i_{\max},j_{\max}} \end{pmatrix}$$

or, more compactly,

$$\frac{d\vec{T}}{dt} + [D_x]\,\vec{T} + [D_y]\,\vec{T} = \vec{S}$$

where $[D_x]$ and $[D_y]$ are matrix representation of the space discretization in the x- and y-directions, respectively.

Implicit Euler Time Advance

If we use the implicit Euler time advance scheme, we get the following fully-discrete form for the problem:

$$\begin{aligned} \frac{\vec{T}^{n+1} - \vec{T}^{n}}{\Delta t} + [D_x]\,\vec{T}^{n+1} + [D_y]\,\vec{T}^{n+1} &= \vec{S} \\ ([I] + \Delta t\,[D_x] + \Delta t\,[D_y])\,\overrightarrow{\delta T} &= -\Delta t\left([D_x]\,\vec{T}^{n} + [D_y]\,\vec{T}^{n}\right) + \Delta t\,\vec{S} \end{aligned} \qquad (8.3)$$

The right-hand side of Equation 8.3 is the flux integral at time level n. The left-hand side is a matrix with structure similar or identical to the system of equations

we would have had to solve for the Laplace equation. That is, the matrix looks roughly like this for a 4-by-4 mesh:

$$\begin{bmatrix}
D & x & & & y & & & & & & & & & & & \\
X & D & x & & & y & & & & & & & & & & \\
 & X & D & x & & & y & & & & & & & & & \\
 & & X & D & & & & y & & & & & & & & \\
Y & & & & D & x & & & y & & & & & & & \\
 & Y & & & X & D & x & & & y & & & & & & \\
 & & Y & & & X & D & x & & & y & & & & & \\
 & & & Y & & & X & D & & & & y & & & & \\
 & & & & Y & & & & D & x & & & y & & & \\
 & & & & & Y & & & X & D & x & & & y & & \\
 & & & & & & Y & & & X & D & & & & y & \\
 & & & & & & & Y & & & & D & x & & & \\
 & & & & & & & & Y & & & X & D & x & & \\
 & & & & & & & & & Y & & & X & D & x & \\
 & & & & & & & & & & Y & & & X & D &
\end{bmatrix} \tag{8.4}$$

This system of equations is impractical to solve directly for the same reasons that we discussed for the Laplace equation. In this case, however, we have an option that was not available to us in that case: approximate factorization. We can re-write the left-hand side of Equation 8.3 as

$$\left([I]+\Delta t\,[D_x]+\Delta t\,[D_y]\right)\overrightarrow{\delta T} \;\approx\; \left([I]+\Delta t\,[D_x]\right)\left([I]+\Delta t\,[D_y]\right)\overrightarrow{\delta T} \tag{8.5}$$

This is a win, because each of $[D_x]$ and $[D_y]$ are tridiagonal.[1] The question is, does this factorization produce an unacceptable error? To find out, expand the approximately factored form:

$$\left([I]+\Delta t\,[D_x]\right)\left([I]+\Delta t\,[D_y]\right)\overrightarrow{\delta T} = \left([I]+\Delta t\,[D_x]+\Delta t\,[D_y]+\Delta t^2\,[D_x]\,[D_y]\right)\overrightarrow{\delta T} \tag{8.6}$$

Comparing Equations 8.5 and 8.6, we see that the approximate factorization introduces an error of $O\left(\Delta t^2\overrightarrow{\delta T}\right) = O\left(\Delta t^3\right)$ (where the equality reflects the fact that $\overrightarrow{\delta T} \sim \Delta t$). But we know that the implicit Euler scheme is only first-order accurate in time, so there is *already* an error of $O\left(\Delta t^2\right)$ *before* we do the approximate factorization. Therefore, the factorization itself does not hurt time accuracy.

[1] $[D_y]$ is only tridiagonal in the sense of having two diagonals adjacent to the main diagonal if we reorder the unknowns in $\vec{T}$ so that the j index varies more rapidly than the i index.

Trapezoidal Time Advance

If instead we use the trapezoidal (centered implicit) time advance scheme, we get the following fully-discrete form for the problem:

$$\begin{aligned} \frac{\vec{T}^{n+1}-\vec{T}^{n}}{\Delta t}+[D_x]\frac{\vec{T}^{n+1}+\vec{T}^{n}}{2}+[D_y]\frac{\vec{T}^{n+1}+\vec{T}^{n}}{2} &= \vec{S} \\ \left([I]+\frac{\Delta t}{2}[D_x]+\frac{\Delta t}{2}[D_y]\right)\overrightarrow{\delta T} &= -\Delta t\left([D_x]\vec{T}^{n}+[D_y]\vec{T}^{n}\right)+\Delta t\vec{S} \end{aligned} \tag{8.7}$$

Approximately factor the LHS of this equation and show that second-order accuracy is still achieved.

8.4.1 Application of Approximate Factorization

To apply approximate factorization in practice, we combine (for example) Equations 8.3 and 8.5:

$$([I]+\Delta t\,[D_x])\,([I]+\Delta t\,[D_y])\,\overrightarrow{\delta T}=-\Delta t\left([D_x]\vec{T}^{n}+[D_y]\vec{T}^{n}\right)+\Delta t\,\vec{S}$$

We can solve this system of equations in two steps:

$$\begin{aligned}([I]+\Delta t\,[D_x])\,\overrightarrow{\delta\vec{T}} &= -\Delta t\left([D_x]\,\vec{T}^n+[D_y]\,\vec{T}^n\right)+\Delta t\,\vec{S}\\ ([I]+\Delta t\,[D_y])\,\overrightarrow{\delta\vec{T}} &= \overrightarrow{\delta\vec{T}}\end{aligned}$$

That is, instead of solving one large matrix problem, we can now solve a series of small matrix problems along lines in the mesh. First, for every line of constant j (lines parallel to the x-axis) we solve for an intermediate variable along that line $\overrightarrow{\delta\vec{T}}_j$:

$$([I]+\Delta t\,[D_x])_j\,\overrightarrow{\delta\vec{T}_j} = -\Delta t\left([D_x]\,\vec{T}^n+[D_y]\,\vec{T}^n\right)_j+\Delta t\,\vec{S} \tag{8.8}$$

Then, for every line of constant i (lines parallel to the y-axis) we solve for the update to the solution along that line $\overrightarrow{\delta\vec{T}}_i$:

$$([I]+\Delta t\,[D_y])_i\,\overrightarrow{\delta\vec{T}_i} = \overrightarrow{\delta\vec{T}_i} \tag{8.9}$$

That all sounds very nice, but it *is* a little abstract. To make it more concrete, consider the case of the two-dimensional heat equation. For this case, we can write Equation 8.3 for the interior control volume $(4,2)$ as:

$$\begin{aligned}\left(1+\frac{2\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}+\frac{2\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)\delta T_{4,2} &\quad -\left(\frac{2\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}+\frac{2\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)T^n_{4,2}\\ +\left(\frac{u\Delta t}{2\Delta x}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}\right)\delta T_{5,2} &\quad -\left(\frac{u\Delta t}{2\Delta x}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}\right)T^n_{5,2}\\ +\left(-\frac{u\Delta t}{2\Delta x}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}\right)\delta T_{3,2} &= -\left(-\frac{u\Delta t}{2\Delta x}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}\right)T^n_{3,2}\\ +\left(\frac{v\Delta t}{2\Delta y}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)\delta T_{4,3} &\quad -\left(\frac{v\Delta t}{2\Delta y}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)T^n_{4,3}\\ +\left(-\frac{v\Delta t}{2\Delta y}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)\delta T_{4,1} &\quad -\left(-\frac{v\Delta t}{2\Delta y}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)T^n_{4,1}\end{aligned}$$

Applying approximate factorization, we have two equations, one for $\delta\tilde{T}_{4,2}$ (solved for simultaneously with $\delta\tilde{T}_{5,2}$ and $\delta\tilde{T}_{3,2}$, etc.) and the other for $\delta T_{4,2}$ (solved for simultaneously with $\delta T_{4,3}$ and $\delta T_{4,1}$, etc).

$$-\left(\frac{2\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}+\frac{2\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)T^n_{4,2}$$

$$
\begin{aligned}
+\left(\frac{u\Delta t}{2\Delta x}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}\right)\delta\tilde{T}_{5,2} \qquad & -\left(\frac{u\Delta t}{2\Delta x}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}\right)T^n_{5,2} \\
\left(1+\frac{2\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}\right)\delta\tilde{T}_{4,2} \;&=\; -\left(-\frac{u\Delta t}{2\Delta x}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}\right)T^n_{3,2} \\
+\left(-\frac{u\Delta t}{2\Delta x}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta x^2}\right)\delta\tilde{T}_{3,2} \qquad & -\left(\frac{v\Delta t}{2\Delta y}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)T^n_{4,3} \\
& -\left(-\frac{v\Delta t}{2\Delta y}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)T^n_{4,1}
\end{aligned}
$$

$$
\begin{aligned}
+\left(\frac{v\Delta t}{2\Delta y}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)\delta T_{4,3} & \\
\left(1+\frac{2\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)\delta T_{4,2} \;&=\; \delta\tilde{T}_{4,2} \\
+\left(-\frac{v\Delta t}{2\Delta y}-\frac{\Delta t}{\mathrm{Re}\cdot\mathrm{Pr}\cdot\Delta y^2}\right)\delta T_{4,1} &
\end{aligned}
$$

These last two equations are specific instances of Equations 8.8 and 8.9.

Let us return now to the full form of these equations. If we completely ignore boundary conditions, then the compact matrix forms of the difference operators are given by:

$$
[D_x]=\frac{1}{Re\cdot Pr\cdot\Delta x^2}\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}+\frac{u}{2\Delta x}\begin{bmatrix} 0 & 1 & & & & \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & -1 & 0 & 1 & \\ & & & -1 & 0 & 1 \\ & & & & -1 & 0 \end{bmatrix}
$$

and

$$
[D_y]=\frac{1}{Re\cdot Pr\cdot\Delta y^2}\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}+\frac{v}{2\Delta y}\begin{bmatrix} 0 & 1 & & & & \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & -1 & 0 & 1 & \\ & & & -1 & 0 & 1 \\ & & & & -1 & 0 \end{bmatrix}
$$

So if we define $\alpha \equiv \frac{1}{Re \cdot Pr}$, Equation 8.8 becomes

$$\begin{bmatrix} 1+\frac{2\alpha\Delta t}{\Delta x^2} & -\frac{\alpha\Delta t}{\Delta x^2}+\frac{u}{2\Delta x} & & & & \\ -\frac{\alpha\Delta t}{\Delta x^2}-\frac{u}{2\Delta x} & 1+\frac{2\alpha\Delta t}{\Delta x^2} & -\frac{\alpha\Delta t}{\Delta x^2}+\frac{u}{2\Delta x} & & & \\ & -\frac{\alpha\Delta t}{\Delta x^2}-\frac{u}{2\Delta x} & 1+\frac{2\alpha\Delta t}{\Delta x^2} & -\frac{\alpha\Delta t}{\Delta x^2}+\frac{u}{2\Delta x} & & \\ & & -\frac{\alpha\Delta t}{\Delta x^2}-\frac{u}{2\Delta x} & 1+\frac{2\alpha\Delta t}{\Delta x^2} & -\frac{\alpha\Delta t}{\Delta x^2}+\frac{u}{2\Delta x} & \\ & & & -\frac{\alpha\Delta t}{\Delta x^2}-\frac{u}{2\Delta x} & 1+\frac{2\alpha\Delta t}{\Delta x^2} & -\frac{\alpha\Delta t}{\Delta x^2}+\frac{u}{2\Delta x} \\ & & & & -\frac{\alpha\Delta t}{\Delta x^2}-\frac{u}{2\Delta x} & 1+\frac{2\alpha\Delta t}{\Delta x^2} \end{bmatrix} \begin{pmatrix} \delta\tilde{T}_{1,j} \\ \delta\tilde{T}_{2,j} \\ \delta\tilde{T}_{3,j} \\ \delta\tilde{T}_{4,j} \\ \delta\tilde{T}_{5,j} \\ \delta\tilde{T}_{6,j} \end{pmatrix} = (\text{RHS})$$

We would solve 6 of these equations (using the Thomas algorithm, described in Appendix C), one for each value of j. Next we apply Equation 8.9, which for this case becomes

$$\begin{bmatrix} 1+\frac{2\alpha\Delta t}{\Delta y^2} & -\frac{\alpha\Delta t}{\Delta y^2}+\frac{v}{2\Delta y} & & & & \\ -\frac{\alpha\Delta t}{\Delta y^2}-\frac{v}{2\Delta y} & 1+\frac{2\alpha\Delta t}{\Delta y^2} & -\frac{\alpha\Delta t}{\Delta y^2}+\frac{v}{2\Delta y} & & & \\ & -\frac{\alpha\Delta t}{\Delta y^2}-\frac{v}{2\Delta y} & 1+\frac{2\alpha\Delta t}{\Delta y^2} & -\frac{\alpha\Delta t}{\Delta y^2}+\frac{v}{2\Delta y} & & \\ & & -\frac{\alpha\Delta t}{\Delta y^2}-\frac{v}{2\Delta y} & 1+\frac{2\alpha\Delta t}{\Delta y^2} & -\frac{\alpha\Delta t}{\Delta y^2}+\frac{v}{2\Delta y} & \\ & & & -\frac{\alpha\Delta t}{\Delta y^2}-\frac{v}{2\Delta y} & 1+\frac{2\alpha\Delta t}{\Delta y^2} & -\frac{\alpha\Delta t}{\Delta y^2}+\frac{v}{2\Delta y} \\ & & & & -\frac{\alpha\Delta t}{\Delta y^2}-\frac{v}{2\Delta y} & 1+\frac{2\alpha\Delta t}{\Delta y^2} \end{bmatrix} \begin{pmatrix} \delta T_{i,1} \\ \delta T_{i,2} \\ \delta T_{i,3} \\ \delta T_{i,4} \\ \delta T_{i,5} \\ \delta T_{i,6} \end{pmatrix} = \begin{pmatrix} \delta\tilde{T}_{i,1} \\ \delta\tilde{T}_{i,2} \\ \delta\tilde{T}_{i,3} \\ \delta\tilde{T}_{i,4} \\ \delta\tilde{T}_{i,5} \\ \delta\tilde{T}_{i,6} \end{pmatrix}$$

We solve this equation 6 times as well, once for each i. This gives us $\delta T_{i,j}$ for all i and j in the mesh. We can update the solution (including possible ghost cells) and continue to the next time step.

8.4.2 Implicit Implementation of Boundary Conditions

With implicit time advance, we need to be able to apply boundary conditions implicitly as well. That is, the changes in solution must be computed so that the boundary conditions are still satisfied at the new time level $n+1$. We will examine both Dirichlet and Neumann boundary conditions in this context. In both cases, we will assume that the boundary condition is satisfied at time level n.

Implicit Dirichlet boundary conditions

Dirichlet boundary conditions are typically enforced using a ghost cell, with the requirement that

$$\frac{\overline{T}_{i,1}+\overline{T}_{i,0}}{2} = T_w$$

We know that this is true at time level n and wish to update the solution so that it will also be true at time level $n+1$. That is,

$$\begin{aligned} \frac{\overline{T}^n_{i,1} + \overline{T}^n_{i,0}}{2} &= T_w \\ \frac{\overline{T}^{n+1}_{i,1} + \overline{T}^{n+1}_{i,0}}{2} &= T_w \end{aligned}$$

If we subtract these two equations, we get

$$\begin{aligned} \frac{\delta\overline{T}_{i,1} + \delta\overline{T}_{i,0}}{2} &= 0 \\ \delta\overline{T}_{i,0} &= -\delta\overline{T}_{i,1} \end{aligned}$$

Implicit Neumann boundary conditions

Neumann boundary conditions are also typically enforced using a ghost cell, with the requirement that

$$\frac{\overline{T}_{i,1} - \overline{T}_{i,0}}{\Delta y} = \frac{\partial T}{\partial y}_w$$

We know that this is true at time level n and wish to update the solution so that it will also be true at time level $n+1$. That is,

$$\begin{aligned} \frac{\overline{T}^{n+1}_{i,1} - \overline{T}^{n+1}_{i,0}}{\Delta y} &= \frac{\partial T}{\partial y}_w \\ \frac{\overline{T}^n_{i,1} - \overline{T}^n_{i,0}}{\Delta y} &= \frac{\partial T}{\partial y}_w \end{aligned}$$

If we subtract these two equations, we get

$$\begin{aligned} \frac{\delta\overline{T}_{i,1} - \delta\overline{T}_{i,0}}{\Delta y} &= 0 \\ \delta\overline{T}_{i,0} &= \delta\overline{T}_{i,1} \end{aligned}$$

Enforcement in conjunction with tri-diagonal solution

These boundary conditions are added to the tri-diagonal system. In fact, boundary conditions of some sort are required to close these systems, which require the ghost

cell control volumes for flux computation but may not yet contain boundary conditions (as written in Equation 8.10, for example). The full and correct version of that equation, with a Dirichlet condition at $i = \frac{1}{2}$ and Neumann at $i = 6\frac{1}{2}$ is in fact:

$$\begin{bmatrix} 1 & 1 & & & & & & & \\ -\frac{\alpha\Delta t}{\Delta x^2} - \frac{u}{2\Delta x} & 1 + \frac{2\alpha\Delta t}{\Delta x^2} & -\frac{\alpha\Delta t}{\Delta x^2} + \frac{u}{2\Delta x} & & & & & & \\ & -\frac{\alpha\Delta t}{\Delta x^2} - \frac{u}{2\Delta x} & 1 + \frac{2\alpha\Delta t}{\Delta x^2} & \ddots & & & & & \\ & & \ddots & \ddots & \ddots & & & & \\ & & & \ddots & \ddots & \ddots & & & \\ & & & & \ddots & 1 + \frac{2\alpha\Delta t}{\Delta x^2} & -\frac{\alpha\Delta t}{\Delta x^2} + \frac{u}{2\Delta x} & & \\ & & & & & -\frac{\alpha\Delta t}{\Delta x^2} - \frac{u}{2\Delta x} & 1 + \frac{2\alpha\Delta t}{\Delta x^2} & -\frac{\alpha\Delta t}{\Delta x^2} + \frac{u}{2\Delta x} \\ & & & & & & 1 & -1 \end{bmatrix} \begin{pmatrix} \delta\tilde{T}_{0,j} \\ \delta\tilde{T}_{1,j} \\ \delta\tilde{T}_{2,j} \\ \delta\tilde{T}_{3,j} \\ \delta\tilde{T}_{4,j} \\ \delta\tilde{T}_{5,j} \\ \delta\tilde{T}_{6,j} \\ \delta\tilde{T}_{7,j} \end{pmatrix} = \begin{pmatrix} 0 \\ \mathrm{RHS}_1 \\ \mathrm{RHS}_2 \\ \mathrm{RHS}_3 \\ \mathrm{RHS}_4 \\ \mathrm{RHS}_5 \\ \mathrm{RHS}_6 \\ 0 \end{pmatrix} \tag{8.11}$$

This enlarged system is solved in the usual way.

Chapter 9

Systems of PDE's

Learning Objectives. Students will be able to:

- Describe the extension of scalar discretization methods to systems of equations.
- Explain why flux Jacobians are a critical part of implicit computation for systems of equations.
- Compute Jacobian of a flux vector with respect to a set of unknowns.

We've talked all term about linear, scalar model equations, because these are easier to analyze and understand than non-linear systems of equations. Now it's time to take the plunge and look at systems of equations. Instead of a single unknown u, we now have a vector of unknowns U at each point. This vector might be as simple $(u, v, w, P)^T$ for the incompressible flow equations or something like $(\rho, \rho u, \rho v, \rho w, E, E_v, \rho_{O_2}, \rho_N, \rho_O, \rho_{NO})^T$ for a compressible reacting air calculation[1], where E_v is the energy in vibrational modes and the subscripted densities are species densities. Or it might be something even more complicated.

The evolution of these unknowns depends on flux vectors F, G, and H, (in the x-, y-, and z-directions, respectively) and on a source vector S, all of which are (possibly non-linear) functions of U.

$$\frac{d\bar{U}_{i,j,k}}{dt} + \oint_{\partial CV} \begin{pmatrix} F\hat{i} \\ G\hat{j} \\ H\hat{k} \end{pmatrix} \cdot \hat{n}\, dA = \bar{S}_{i,j,k} \tag{9.1}$$

[1] A model like this might be used for earth re-entry problems, for instance.

For the incompressible flow case above, the flux vector F might be $(u^2+P,\, uv,\, uw,\, u/\beta)^T$; this corresponds to a method known as *artificial compressibility*. For an inviscid reacting air problem, the flux vector would be $\left(\rho u,\, \rho u^2+P,\, \rho uv,\, \rho uw,\, u(E+P),\, uE_v,\, \rho_{O_2}u,\, \rho_N u,\, \rho_O\right.$

Systems of equations are discretized in much the same way as scalar equations. For example, we can discretize the one-dimensional equivalent of Equation 9.1 as

$$\frac{U_i^{n+1}-U_i^n}{\Delta t} = -\frac{F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n}{\Delta x} + S_i^n \tag{9.2}$$

Fluxes at $i+\frac{1}{2}$ can be calculated using the same approaches we've already discussed.

This discretization uses the explicit Euler time advance scheme. From our study of model equations, we know the stability region for this time advance scheme. To prove (or disprove) the stability of a linear system of PDE's, "all" we would need to do is find the eigenvalues of the system of equations implied by Equation 9.2; this would lead us to a stability bound just as similar analysis did for the scalar case. For non-linear systems, things are more complicated; a good general rule is that the stability bound will be lower for non-linear problems than for the linearization of the same non-linear problem.

The explicit case is simple to implement; all that need be done is to compute the fluxes and take differences of them. However, for systems just as for scalar equations, explicit schemes often prove to be inefficient. A simple implicit discretization of the conservation law would be

$$\frac{U_i^{n+1}-U_i^n}{\Delta t} = -\frac{F_{i+\frac{1}{2}}^{n+1} - F_{i-\frac{1}{2}}^{n+1}}{\Delta x} + S_i^{n+1} \tag{9.3}$$

The catch is that $F_{i+\frac{1}{2}}^{n+1}$ is a function of (for example) U_i^{n+1} and U_{i+1}^{n+1}. However, we can approximate

$$F_{i+\frac{1}{2}}^{n+1} \equiv F\left(U_i, U_{i+1}\right)^{n+1} \approx F\left(U_i, U_{i+1}\right)^n + \Delta t \left.\frac{\partial F\left(U_i, U_{i+1}\right)}{\partial t}\right|^n + O\left(\Delta t^2\right)$$

and

$$\left.\frac{\partial F\left(U_i, U_{i+1}\right)}{\partial t}\right|^n = \left.\frac{\partial F}{\partial U_i}\frac{\partial U_i}{\partial t}\right|^n + \left.\frac{\partial F}{\partial U_{i+1}}\frac{\partial U_{i+1}}{\partial t}\right|^n \tag{9.4}$$

by the chain rule.

We can re-write Equation 9.3 in δ-form as

$$\left(\frac{I}{\Delta t}+\frac{1}{\Delta x}\left.\frac{\partial F(U_i,U_{i+1})}{\partial U_i}\right|^n-\frac{1}{\Delta x}\left.\frac{\partial F(U_{i-1},U_i)}{\partial U_i}\right|^n-\left.\frac{\partial S}{\partial U}\right|_i^n\right)\delta U_i^{n+1}$$
$$+\frac{1}{\Delta x}\left.\frac{\partial F(U_i,U_{i+1})}{\partial U_{i+1}}\right|^n\delta U_{i+1}^{n+1}-\frac{1}{\Delta x}\left.\frac{\partial F(U_{i-1},U_i)}{\partial U_{i-1}}\right|^n\delta U_{i-1}^{n+1} = -\frac{F_{i+\frac{1}{2}}^n-F_{i-\frac{1}{2}}^n}{\Delta x}+S_i^n \quad (9.5)$$

In hope of shedding some light on this, let's look at a specific example where F and U are both scalars. For the energy equation, $U=T$ and the flux can be written as $F=u\frac{T_i+T_{i+1}}{2}-\alpha\frac{T_{i+1}-T_i}{\Delta x}$, using centered evaluation of the advective flux. Then Equation 9.4 can be written for this case as:

$$\left.\frac{\partial F\left(U_i,U_{i+1}\right)}{\partial t}\right|^n=\left(\frac{u}{2}+\frac{\alpha}{\Delta x}\right)^n\frac{\partial T_i^n}{\partial t}+\left(\frac{u}{2}-\frac{\alpha}{\Delta x}\right)^n\frac{\partial T_{i+1}^n}{\partial t}$$

and the fully implicit discretization can be written as

$$\left(\frac{1}{\Delta t}+\frac{1}{\Delta x}\left(\frac{u}{2}+\frac{\alpha}{\Delta x}\right)-\frac{1}{\Delta x}\left(\frac{u}{2}-\frac{\alpha}{\Delta x}\right)-\left.\frac{\partial S}{\partial U}\right|_i^n\right)\delta U_i^{n+1}$$
$$+\frac{1}{\Delta x}\left(\frac{u}{2}+\frac{\alpha}{\Delta x}\right)\delta U_{i+1}^{n+1}-\frac{1}{\Delta x}\left(\frac{u}{2}-\frac{\alpha}{\Delta x}\right)\delta U_{i-1}^{n+1} = -\frac{F_{i+\frac{1}{2}}^n-F_{i-\frac{1}{2}}^n}{\Delta x}+S_i^n$$

This is more or less the result we expected.

There are two major things that we need to know before we can solve the system of equations posed by Equation 9.5: how to compute the *Jacobians* $\frac{\partial F}{\partial U}$ and $\frac{\partial S}{\partial U}$, and how to solve the algebraic equations that arise.

9.1 Computation of Flux and Source Jacobians

So we have a vector function $F(U_i,U_{i+1})$ and we need to know the partial derivatives of the components of F with respect to the components of U_i. For this purpose, we can treat the components of $\underline{U_{i+1}}$ as constants and write simply $F=F(U)$. In general, all components of the flux F depend on all components of the state U. That

is,

$$F(U) = \begin{pmatrix} F_1(U_1, U_2, U_3, \ldots, U_m) \\ F_2(U_1, U_2, U_3, \ldots, U_m) \\ F_3(U_1, U_2, U_3, \ldots, U_m) \\ \vdots \\ F_m(U_1, U_2, U_3, \ldots, U_m) \end{pmatrix}$$

One must always write the components of F explicitly in terms of components of U before computing

$$\frac{\partial F}{\partial U} = \begin{pmatrix} \frac{\partial F_1}{\partial U_1} & \frac{\partial F_1}{\partial U_2} & \frac{\partial F_1}{\partial U_3} & \cdots & \frac{\partial F_1}{\partial U_m} \\ \frac{\partial F_2}{\partial U_1} & \frac{\partial F_2}{\partial U_2} & \frac{\partial F_2}{\partial U_3} & \cdots & \frac{\partial F_2}{\partial U_m} \\ \frac{\partial F_3}{\partial U_1} & \frac{\partial F_3}{\partial U_2} & \frac{\partial F_3}{\partial U_3} & \cdots & \frac{\partial F_3}{\partial U_m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial U_1} & \frac{\partial F_m}{\partial U_2} & \frac{\partial F_m}{\partial U_3} & \cdots & \frac{\partial F_m}{\partial U_m} \end{pmatrix}$$

And that, believe it or not, is that. Note that the subscripts in these equations represent the component of the flux vector F or the vector of unknowns U, not anything do to with the spatial discretization.

9.1.1 Example: Nearly the compressible Euler equations

Consider the following very simple case (compressible mass and momentum conservation with no pressure term):

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \end{pmatrix} \qquad F = \begin{pmatrix} \rho u \\ \rho u^2 \\ \rho uv \end{pmatrix}$$

The first step is to re-write components of F in terms of components of U:

$$F = \begin{pmatrix} \rho u \\ \rho u^2 \\ \rho uv \end{pmatrix} = \begin{pmatrix} U_2 \\ \frac{U_2^2}{U_1} \\ \frac{U_2 U_3}{U_1} \end{pmatrix}$$

From here on things are straightforward:

$$\frac{\partial F}{\partial U} = \begin{bmatrix} 0 & 1 & 0 \\ -\frac{U_2^2}{U_1^2} & 2\frac{U_2}{U_1} & 0 \\ -\frac{U_2 U_3}{U_1^2} & \frac{U_3}{U_1} & \frac{U_2}{U_1} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -u^2 & 2u & 0 \\ -uv & v & u \end{bmatrix}$$

9.1.2 Example: the compressible Euler equations

Compute the Jacobian for the following case:

$$U = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} \qquad F = \begin{pmatrix} \rho u \\ \rho u^2 + P \\ u(E+P) \end{pmatrix}$$

where $P = \left(E - \frac{1}{2}\rho u^2\right)(\gamma - 1)$. Remember to first write components of F in terms of components of U.

9.2 Problems

1. Source Jacobian for One-Dimensional Flow of Dissociating Oxygen Consider the case of one-dimensional compressible flow of oxygen with dissociation but no ionization. The species present in the flow are O_2 and O. The governing equation for this problem is

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = S$$

where

$$U = \begin{pmatrix} \rho \\ \rho u \\ E \\ \rho_O \end{pmatrix}$$

$$F = \begin{pmatrix} \rho u \\ \rho u^2 + P \\ u(E+P) \\ u\rho_O \end{pmatrix}$$

and

$$S = \begin{pmatrix} 0 \\ 0 \\ w_O \left(H^0_{\rho_O} - RT \right) \\ w_O \end{pmatrix}$$

Several notes are in order.

- Don't worry too much about understanding the physics of this problem, for two reasons. First, the physics is grossly over-simplified from the real world and therefore isn't worth a huge amount of effort. Second, the problem can be done by manipulating things algebraically without knowledge of the underlying physics.
- The energy E is defined as:

$$E = \left(\frac{3}{2} RT + H^0_{\rho_O} \right) \rho_O + \frac{5}{2} RT \rho_{O_2} + \frac{\rho u^2}{2}$$

  where $H^0_{\rho_O}$ is the chemical heat of formation of monatomic oxygen. The internal energy expressions for O and O_2 differ because O_2, as a diatomic molecule, has rotational and vibrational energy modes which O does not have.

- The pressure P is defined as $P = \rho RT$.
- The mass density of O_2 is not solved for explicitly, but instead is represented as $\rho_{O_2} = \rho - \rho_O$. The choice of computing ρ_{O_2} by subtraction is made because the mass fraction of O_2 is expected to be close to one; subtracting in this way reduces round-off errors.
- The creation of monatomic oxygen is given by

$$w_O = (\rho_{O_2} - \rho_O) A \exp(-BT)$$

- Clearly it would be a lot easier to compute Jacobians using temperature instead of energy as a dependent variable. That is, using

$$V = \begin{pmatrix} \rho \\ u \\ T \\ \rho_O \end{pmatrix}$$

in place of U.

Find the flux and source Jacobians ($\frac{\partial F}{\partial U}$ and $\frac{\partial S}{\partial U}$, respectively). You will probably want to take advantage of the chain rule. If you know $\frac{\partial U}{\partial V}$ and $\frac{\partial S}{\partial V}$, how can you find $\frac{\partial S}{\partial U}$ without analytically deriving it? (This is often a useful trick, as some variable transformations have treacherous Jacobians in one direction but not the other.

Chapter 10

The Incompressible Navier-Stokes Equations

The incompressible Navier-Stokes equations are mathematically rich — they are a coupled system of non-linear elliptic PDE's. There are a reasonably large number of similarity transformations that can be usefully applied to these equations, a topic on which entire monographs are written. Furthermore, the continuity equation is physically not an evolution equation but a partial differential constraint on the velocity field.

This combination of attributes, while fascinating from the point of view of applied mathematics, makes the incompressible Navier-Stokes equations difficult to solve numerically. There are a number of approaches to solving this system, each characterized by the way in which the continuity equation is satisfied.

- Stream function-vorticity methods re-write the velocity field in terms of derivatives of the stream function. The conservation equation is satisfied automatically because of the relationship between stream function and velocity. The momentum equations are replaced by a single vorticity transport equation, and a Poisson equation relates the stream function and vorticity. This method works fine for two-dimensional flows, but can not be applied easily in three dimensions, where there is no simple scalar analog to the stream function.

- Pressure Poisson methods take the divergence of the momentum equations and manipulate this to get a Poisson equation for the pressure. This equation can be simplified by using the continuity equation to eliminate terms.

Then the momentum equations are advanced one time step at a time, with fixed pressure for a single time step. Between time steps, the pressure is re-computed by using the Poisson equation, whose source term depends on derivatives of the velocity. These methods are well-established in the field, but tend to be tricky to program, especially with respect to getting the pressure Poisson equation coupled correctly to momentum.

- Artificial compressibility methods add an aphysical time derivative of pressure to the continuity equation. This derivative is scaled by a parameter β that effectively sets the pseudo-compressibility of the fluid. The addition of a time derivative of pressure couples the continuity equation more tightly to the momentum equations and allows us to advance pressure and velocity in time together. Note that some additional effort is required to make this approach time accurate and that the choice of β can, in some circumstances, affect the final solution of a problem. The non-dimensional Navier-Stokes equations in artificial compressibility form can be written as

$$\begin{aligned} \frac{\partial P}{\partial t} + \frac{1}{\beta}\frac{\partial u}{\partial x} + \frac{1}{\beta}\frac{\partial v}{\partial y} &= 0 \\ \frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} &= -\frac{\partial P}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \\ \frac{\partial v}{\partial t} + \frac{\partial uv}{\partial x} + \frac{\partial v^2}{\partial y} &= -\frac{\partial P}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \end{aligned} \tag{10.1}$$

10.1 Discretization

It is a simple matter to re-write Equation 10.1 as

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0 \tag{10.2}$$

where

$$U = \begin{pmatrix} P \\ u \\ v \end{pmatrix} \qquad F = \begin{pmatrix} \frac{u}{\beta} \\ u^2 + P - \frac{1}{Re}\frac{\partial u}{\partial x} \\ uv - \frac{1}{Re}\frac{\partial v}{\partial x} \end{pmatrix} \qquad G = \begin{pmatrix} \frac{v}{\beta} \\ uv - \frac{1}{Re}\frac{\partial u}{\partial y} \\ v^2 + P - \frac{1}{Re}\frac{\partial v}{\partial y} \end{pmatrix}$$

Using our traditional magic with integration over a computational cell; applying Gauss's Theorem; and dividing by the size of the finite volume gives

$$\frac{dU_{i,j}}{dt} + \frac{F_{i+\frac{1}{2},j} - F_{i-\frac{1}{2},j}}{\Delta x} + \frac{G_{i,j+\frac{1}{2}} - G_{i,j-\frac{1}{2}}}{\Delta y} = 0 \tag{10.3}$$

where U, F, and G have the same definitions as before. That part of the derivation is a simple application of mathematics that has been done elsewhere in these notes in several places.

The key to reducing Equation 10.3 to a fully-discrete formulation that we can program is deciding how to evaluate fluxes and what time advance scheme to use. As an illustration and for the project for this course, we will use centered flux evalua-

tions and implicit Euler time advance. The sole remaining difficulty is that we do not have data at time level $n+1$ to evaluate the fluxes on the right-hand side. We can, however, use Taylor series expansions to write these fluxes in terms of data at time level n. For example,

$$\begin{aligned} F^{n+1}_{i+\frac{1}{2},j} &= F\left(U^{n+1}_{i,j}, U^{n+1}_{i+1,j}\right) \\ &= F\left(U^{n}_{i,j} + \delta U_{i,j}, U^{n}_{i+1,j} + \delta U_{i+1,j}\right) \end{aligned} \tag{10.4}$$

$$= F\left(U^n_{i,j}, U^n_{i+1,j}\right) + \frac{\partial F^n_{i+\frac{1}{2},j}}{\partial U_{i,j}}\delta U_{i,j} + \frac{\partial F^n_{i+\frac{1}{2},j}}{\partial U_{i+1,j}}\delta U_{i+1,j} + O\left((\delta U)^2\right) \quad (10.5)$$

where a two-variable Taylor series expansion is used between Equations 10.4 and 10.5.

This result requires that we calculate flux Jacobians: derivatives of the fluxes with respect to the unknowns in nearby control volumes; see Section 9.1 for more information about how to do this. The Jacobians we obtain are:

$$\frac{\partial F^n_{i+\frac{1}{2},j}}{\partial U_{i,j}} = \begin{bmatrix} 0 & \frac{1}{2\beta} & 0 \\ \frac{1}{2} & \frac{u_{i,j}+u_{i+1,j}}{2} + \frac{1}{\Delta x \cdot Re} & 0 \\ 0 & \frac{v_{i,j}+v_{i+1,j}}{4} & \frac{u_{i,j}+u_{i+1,j}}{4} + \frac{1}{\Delta x \cdot Re} \end{bmatrix}$$

$$\frac{\partial F^n_{i+\frac{1}{2},j}}{\partial U_{i+1,j}} = \begin{bmatrix} \; \end{bmatrix}$$

$$\frac{\partial G^n_{i,j+\frac{1}{2}}}{\partial U_{i,j}} = \begin{bmatrix} \; \end{bmatrix}$$

$$\frac{\partial G^n_{i,j+\frac{1}{2}}}{\partial U_{i,j+1}} = \begin{bmatrix} \; \end{bmatrix}$$

where the inviscid and viscous parts have been combined.

If we substitute the expanded fluxes into the fully-discrete equation, we get:

$$\begin{aligned}\left(\frac{I}{\Delta t}+\frac{1}{\Delta x}\frac{\partial F^n_{i+\frac{1}{2},j}}{\partial U_{i,j}}-\frac{1}{\Delta x}\frac{\partial F^n_{i-\frac{1}{2},j}}{\partial U_{i,j}}+\frac{1}{\Delta y}\frac{\partial G^n_{i,j+\frac{1}{2}}}{\partial U_{i,j}}-\frac{1}{\Delta y}\frac{\partial G^n_{i,j-\frac{1}{2}}}{\partial U_{i,j}}\right)\delta U_{i,j} & \\ +\frac{1}{\Delta x}\frac{\partial F^n_{i+\frac{1}{2},j}}{\partial U_{i+1,j}}\delta U_{i+1,j}-\frac{1}{\Delta x}\frac{\partial F^n_{i-\frac{1}{2},j}}{\partial U_{i-1,j}}\delta U_{i-1,j} &= -\frac{F^n_{i+\frac{1}{2},j}-F^n_{i-\frac{1}{2},j}}{\Delta x} \\ +\frac{1}{\Delta y}\frac{\partial G^n_{i,j+\frac{1}{2}}}{\partial U_{i,j+1}}\delta U_{i,j+1}-\frac{1}{\Delta y}\frac{\partial G^n_{i,j-\frac{1}{2}}}{\partial U_{i,j-1}}\delta U_{i,j-1} & -\frac{G^n_{i,j+\frac{1}{2}}-G^n_{i,j-\frac{1}{2}}}{\Delta y}\end{aligned}$$

Multiplying by Δt and labeling terms in an obvious way, we can write:

$$\begin{aligned}(I+\Delta t\,B_x+\Delta t\,B_y)\,\delta U_{i,j} & \\ +\Delta t\,C_x\,\delta U_{i+1,j}+\Delta t\,A_x\,\delta U_{i-1,j} &= -\Delta t\frac{F^n_{i+\frac{1}{2},j}-F^n_{i-\frac{1}{2},j}}{\Delta x} \\ +\Delta t\,C_y\,\delta U_{i,j+1}+\Delta t\,A_y\,\delta U_{i,j-1} & -\Delta t\frac{G^n_{i,j+\frac{1}{2}}-G^n_{i,j-\frac{1}{2}}}{\Delta y}\end{aligned} \tag{10.6}$$

10.2 Approximate Factorization

We can re-write Equation 10.6 in a similar form to the one we used for the incompressible energy equation by combining the various small matrices (A, B, and C) into large matrices:

$$(I+\Delta t D_x+\Delta t D_y)\begin{pmatrix}\delta U_{1,1}\\ \delta U_{2,1}\\ \vdots\\ \delta U_{i-1,j}\\ \delta U_{i,j}\\ \delta U_{i+1,j}\\ \vdots\\ \delta U_{i_{\max}-1,j_{\max}}\\ \delta U_{i_{\max},j_{\max}}\end{pmatrix}=-\Delta t\left(\frac{F^n_{i+\frac{1}{2},j}-F^n_{i-\frac{1}{2},j}}{\Delta x}+\frac{G^n_{i,j+\frac{1}{2}}-G^n_{i,j-\frac{1}{2}}}{\Delta y}\right)$$

$$
D_x = \begin{bmatrix}
B_{x;1,1} & C_{x;1,1} & & & & & & & & \\
A_{x;2,1} & B_{x;2,1} & C_{x;2,1} & & & & & & & \\
 & \ddots & \ddots & \ddots & & & & & & \\
 & & A_{x;i-1,j} & B_{x;i-1,j} & C_{x;i-1,j} & & & & & \\
 & & & A_{x;i,j} & B_{x;i,j} & C_{x;i,j} & & & & \\
 & & & & A_{x;i,j+1} & B_{x;i,j+1} & C_{x;i,j+1} & & & \\
 & & & & & \ddots & \ddots & \ddots & & \\
 & & & & & & A_{x;I-1,J} & B_{x;I-1,J} & C_{x;I-1,J} \\
 & & & & & & & A_{x;I,J} & B_{x;I,J}
\end{bmatrix}
$$

$$
D_y = \begin{bmatrix}
B_{y;1,1} & & & C_{y;1,1} & & & & & & \\
 & B_{y;2,1} & & & C_{y;2,1} & & & & & \\
 & & \ddots & & & \ddots & & & & \\
A_{y;i-1,j} & & & B_{y;i-1,j} & & & C_{y;i-1,j} & & & \\
 & A_{y;i,j} & & & B_{y;i,j} & & & C_{y;i,j} & & \\
 & & A_{y;i,j+1} & & & B_{y;i,j+1} & & & C_{y;i,j+1} & \\
 & & & \ddots & & & \ddots & \ddots & & \\
 & & & & A_{y;I-1,J} & & & B_{y;I-1,J} & C_{y;I-1,J} & \\
 & & & & & A_{y;I,J} & & & B_{y;I,J}
\end{bmatrix}
$$

We can apply approximate factorization to get:

$$
[I + \Delta t\, D_x]\,[I + \Delta t\, D_y]\,\overrightarrow{\delta U} = -\left(\frac{F^n_{i+\frac{1}{2},j} - F^n_{i-\frac{1}{2},j}}{\Delta x} + \frac{G^n_{i,j+\frac{1}{2}} - G^n_{i,j-\frac{1}{2}}}{\Delta y} \right) \quad (10.7)
$$

This approximately factored system can be solved using the same approach that we discussed for the energy equation: breaking the problem into two sets of line problems. In this case, each line problem is a 3×3 block tridiagonal problem. The extension of the Thomas algorithm to systems is straightforward; see Section C.2.

10.3 Boundary Conditions

Mathematically, the incompressible Navier-Stokes equations are elliptic. Physically, this implies that a change in the boundary conditions on velocity or pressure anywhere in the flow field have an effect on all parts of the flow. Numerically, we

must specify a boundary condition for each velocity component and for pressure at all boundaries for the problem to be well-posed. This section discusses boundary conditions for walls — stationary, moving, and porous — and for inflow and outflow boundaries.

10.3.1 Wall boundaries

We will consider wall boundaries at the top and bottom of a domain ($j = \frac{1}{2}$, $j = j_{\max} + \frac{1}{2}$).

The most important feature of walls in viscous flow is that the velocity difference between the wall and the fluid adjacent to the wall is zero. It is less clear what the correct boundary condition is for pressure. For this, we evaluate the equation for momentum normal to the wall at the wall:

$$\frac{\partial v}{\partial t} + \frac{\partial (uv)}{\partial x} + \frac{\partial \left(v^2\right)}{\partial y} = -\frac{\partial P}{\partial y} + \frac{1}{\mathrm{Re}} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right)$$

Because u and v are zero on the wall for all x and t, the first two terms drop out, as does one term in the Laplacian, leaving

$$\frac{\partial \left(v^2\right)}{\partial y} = -\frac{\partial P}{\partial y} + \frac{1}{\mathrm{Re}} \frac{\partial^2 v}{\partial y^2}$$

Because v is zero at the wall, we can approximate v by a Taylor series expansion with no constant term:

$$v \approx y \left. \frac{\partial v}{\partial y} \right|_0 + \frac{y^2}{2} \frac{\partial^2 v}{\partial y^2} + \cdots$$

Therefore,

$$\frac{\partial \left(v^2\right)}{\partial y} \approx 2y \left(\frac{\partial v}{\partial y} \right)^2_{y=0} + O\left(y^2\right)$$

which is zero at the wall. Also, $\frac{\partial v}{\partial y} = -\frac{\partial u}{\partial x} = 0$ at the wall. While $\frac{\partial^2 v}{\partial y^2}$ is not necessarily zero, in practice it is typically very small, especially for straight walls, so we can use

$$\frac{\partial P}{\partial y} \approx 0$$

as an excellent approximation for the pressure boundary condition at the wall.

10.3.2 Stationary walls

For stationary walls, both the normal and tangential velocity are exactly zero. This means that, for a wall at $j = \frac{1}{2}$, we can set values in ghost cells using

$$\begin{aligned} u_{i,0} &= -u_{i,1} \\ v_{i,0} &= -v_{i,1} \\ P_{i,0} &= +P_{i,1} \end{aligned} \tag{10.8}$$

10.3.3 Moving walls

For moving walls, the normal velocity is zero and the tangential velocity matches the wall velocity. This means that, for a wall at $j = \frac{1}{2}$, we can set values in ghost cells using

$$\begin{aligned} u_{i,0} &= 2u_{\text{wall}} - u_{i,1} \\ v_{i,0} &= -v_{i,1} \\ P_{i,0} &= +P_{i,1} \end{aligned} \tag{10.9}$$

10.3.4 Porous walls

For porous walls, we will assume that the tangential velocity is zero and the normal velocity is given. This means that, for a wall at $j = \frac{1}{2}$, we can set values in ghost cells using

$$\begin{aligned} u_{i,0} &= -u_{i,1} \\ v_{i,0} &= 2v(x,0) - v_{i,1} \\ P_{i,0} &= +P_{i,1} \end{aligned} \tag{10.10}$$

10.3.5 Implicit boundary condition implementation

If we construct the tri-diagonal system for the i^{th} column, we need to have $\delta U_{i,0}$ and $\delta U_{i,j_{\max}+1}$. The implicit BC implementation can be used to relate these quantities to $\delta U_{i,1}$ and $\delta U_{i,j_{\max}}$, respectively. As with the energy equation, we can use this information analytically to eliminate the extra unknowns (resulting in a system of $j_{\max}$ block equations) or we can add extra rows to the system of equations (resulting in a system of $j_{\max+2}$ block equations).

10.3.6 Inflow boundaries

For inflow boundaries, we can specify the flow velocity u_{in}, v_{in}. The pressure gradient will again be given by momentum considerations under the assumption that the flow is fully-developed at the entrance. We will assume that inflow is from the left of the domain, so the x-momentum equation is the relevant one:

$$\frac{\partial u}{\partial t} + \frac{\partial \left(u^2\right)}{\partial x} + \frac{\partial \left(uv\right)}{\partial y} = -\frac{\partial P}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

Because the flow is fully developed, the cross-flow velocity v is zero. Also, variation of u in the stream-wise direction can be neglected for this case. Then

$$\frac{\partial P}{\partial x}_{in} = \frac{1}{Re}\frac{\partial^2 u}{\partial y^2}$$

The right-hand side of this expression can be evaluated either using the prescribed boundary values for u or the interior values at $i = 1$. The former is actually more convenient, especially for implicit application of boundary conditions, because this eliminates the dependence of the ghost cell pressure on interior velocities. In either case, we have

$$u_{0,j} \quad = \quad 2u_{in} - u_{1,j}$$

$$\begin{aligned} v_{0,j} &= -v_{1,j} \\ P_{0,j} &= P_{1,j} - \Delta x \frac{\partial P}{\partial x}_{in} \end{aligned} \tag{10.11}$$

For implicit boundary condition application and assuming that $P_{0.j}$ is independent of u, we can derive expressions relating the change in values in the ghost cells to those in the interior as we did for the wall boundary cases.

10.3.7 Outflow boundaries

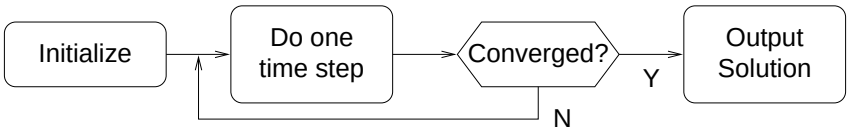
For outflow boundaries, we can specify the pressure P_{out} and assume fully-developed flow (no stream-wise velocity gradient). These conditions are actually quite reasonable. We often want to control the back pressure for internal flows, and a flow that is not fully-developed at the exit may in fact be incorrect computationally, because the boundary may be affecting the solution in the interior rather than simply allowing fluid to leave the domain.

These conditions imply that

$$\begin{aligned} u_{i_{\max}+1,j} &= u_{i_{\max},j} \\ v_{i_{\max}+1,j} &= v_{i_{\max},j} \\ P_{i_{\max}+1,j} &= 2P_{out} - P_{i_{\max},j} \end{aligned} \tag{10.12}$$

The implicit version of Equation 10.12 is

10.4 Outline of Navier-Stokes Code



Initialize

```
Set boundary conditions
```

```
Set parameters (Re, Δt, etc)
Set geometry
Set initial condition
Set ghost cell values
```

Do one time step

```
Compute flux integral everywhere

for each j        ! Line solves along j-lines
  Set up LHS for current j
  Set up implicit BC for current j
  Do block tri-diagonal solution
  ! Can overwrite flux integral with result
end for

for each i        ! Line solves along i-lines
  Set up LHS for current i
  Set up implicit BC for current i
  Do block tri-diagonal solution
  ! Can overwrite flux integral with result
end for

Compute norms of change in solution and update solution
   (including optional over-relaxation)

Set ghost cell values
```

Appendix A

Glossary

amplification factor The ratio of solution magnitude at consecutive time steps as a function of wave number (equivalently, wave length). Used in determining stability of fully-discrete schemes for PDE's. Derived with the assumptions of linearity and periodic boundary conditions.

banded periodic matrix A matrix of size N whose entries along each diagonal with $(i+j)\,\%N$ constant are the same. Such a matrix arising naturally from the spatial discretization of a PDE with periodic boundary conditions.

basis function In the finite-element method, a function defined within the vicinity of a given vertex in the mesh, nearly always with a value of one at that vertex and zero at all other vertices. To determine the value of the solution for a finite element problem at a given location, one sums the contribution from all basis functions at that location.

CFL number A non-dimensional measure of time step, generally used for convective problems and defined as $\text{CFL} = \frac{u\Delta t}{\Delta x}$.

conservation of difficulty A law that states that, given two different ways of doing something, each is equally difficult. The only known exceptions are the result of someone applying Extreme Cleverness to make one of the options simpler.

convergence 1. Obtaining a single numerical solution that is an exact, steady solution to the fully-discretized system of equations under study. 2. The final result of a mesh refinement study: a solution which is for practical purposes free from discretization error.

debugging Finding and eradicating all errors in your program. Bug hunts typically begin with a failed validation or verification test case. See also validation and verification.

discretization Converting a PDE into a coupled system of algebraic equations for the unknowns at particular points in space and possibly time. Also, the system of algebraic equations itself.

discretization error The error introduced by approximating a continuous solution by a finite, discrete set of values.

explicit A time advance scheme in which no data at the new time level is required to advance the solution in time.

fully-discrete form A discretization of a PDE in both time and space.

ghost cell A fictitious cell added outside the computational domain for ease in applying boundary conditions.

implicit A time advance scheme in which data at the new time level *is* required to advance the solution in time.

linear A PDE is said to be linear if the coefficients of the PDE do not depend on the solution of that PDE.

mesh refinement study Determining, by use of successively finer meshes, whether the discretization error in a numerical solution is acceptably small.

modeling Deciding how to mathematically represent the physics of a problem "just simply enough", so that the mathematical representation gives physically realistic solutions without requiring excessive computer resources.

periodic boundary conditions Boundary conditions that enforce periodicity on the solution by requiring that flux leaving one side of the computational domain immediately re-enter on the opposite side.

semi-discrete form A discretization of a PDE in space only. Very useful for analysis, but not typically applied numerically.

stability A numerical scheme is said to be stable if the solution does not grow without bounds.

stationary A PDE is said to be stationary if its coefficients do not vary in time and space.

test function In the finite element method, a local function with properties similar to the basis functions, used to (analytically) multiply the solution en route to discretizing the governing equations.

time accurate A scheme is said to be time accurate if the unsteady solutions it produces are at least first-order accurate in time. For problems where we are interested only in the steady-state solution, time accurate methods are often not a good choice.

validation Selecting, running, and interpreting the results of a series of test cases to demonstrate that the physical models in a program are adequate for a problem or class of problems. In other words, did we code the right set of PDE's? See also debugging and verification.

verification Selecting, running, and interpreting the results of a series of test cases to demonstrate that a program correctly implements the features in its design — i.e., confirming the correctness of a program. In other words, does the code correctly solve the PDE's? See also debugging and validation.

Appendix B

Some Mathematical Concepts Useful for CFD

B.1 Classification of PDE's

Second-degree partial differential equations — those whose highest derivative is a second derivative — are traditionally classified as elliptic, parabolic, and hyperbolic, just as conic sections are in analytic geometry. Any second-order linear PDE with constant coefficients in two dimensions can be written as

$$A\frac{\partial^2 T}{\partial x^2} + B\frac{\partial^2 T}{\partial x \partial y} + C\frac{\partial^2 T}{\partial y^2} = f\left(\frac{\partial T}{\partial x}, \frac{\partial T}{\partial y}, T, x, y\right)$$

This PDE is considered to be

$$\begin{array}{ccc} \text{elliptic} & & < 0 \\ \text{parabolic} & \text{if } B^2 - 4AC & = 0 \\ \text{hyperbolic} & & > 0 \end{array}$$

B.2 Taylor Series Expansions

A smooth function in one dimension can be expanded about the point x_0 into a Taylor series as follows:

$$T(x) = T(x_0) + \frac{dT}{dx}(x_0)(x - x_0) + \frac{d^2T}{dx^2}(x_0)\frac{(x-x_0)^2}{2} + \frac{d^3T}{dx^3}(x_0)\frac{(x-x_0)^3}{6} + \cdots$$

$$+\frac{d^k T}{dx^k}(x_0)\frac{(x-x_0)^k}{k!}+O\left((x-x_0)^{k+1}\right)\cdots$$

A similar expansion can be written in two dimensions:

$$\begin{aligned}T(x,y) &= T(x_0,y_0)+\frac{\partial T}{\partial x}(x_0,y_0)(x-x_0)+\frac{\partial T}{\partial y}(x_0,y_0)(y-y_0)\\ &\quad+\frac{\partial^2 T}{\partial x^2}(x_0,y_0)\frac{(x-x_0)^2}{2}+\frac{\partial^2 T}{\partial x\partial y}(x_0,y_0)(x-x_0)(y-y_0)+\frac{\partial^2 T}{\partial x^2}(x_0,y_0)\frac{(y-y_0)^2}{2}\\ &\quad+\cdots+\sum_{j=0}^{k}\left\{\frac{\partial^k T}{\partial x^{k-j}\partial y^j}(x_0,y_0)\frac{(k-j)!j!}{k!}(x-x_0)^{k-j}(y-y_0)^j\right\}+O\left(\Delta x^{k+1}\right)\end{aligned}$$

B.3 Eigenvalues, Eigenvectors, and All That

Eigensystems appear in several places in CFD analysis, including stability analysis, analysis of PDE systems with multiple unknowns, and analysis of iterative methods for solving large systems of linear equations.

Learning Objectives. Students will be able to:

- Define eigenvalue and eigenvector.
- Describe how to diagonalize the system of coupled ODE's arising from a one-dimensional periodic spatial discretization of a time-dependent PDE to obtain a system of uncoupled ODE's.

B.3.1 Basics about eigensystems

This section is intended to give without proof some basic facts about eigenvalues and eigenvectors of matrices.

The right eigenvectors X_k and the eigenvalues λ_k of a square matrix M of size k are defined as follows:

$$MX_k = \lambda_k X_k \tag{B.1}$$

This system of equations gives each eigenvector to within a constant factor; clearly, each eigenvector can be multiplied by a constant and still satisfy Equation B.1. Similarly, one can define left eigenvectors (which are row vectors) by

$$Y_k M = Y_k \lambda_k \tag{B.2}$$

where the λ_k are the same in each case. The eigensystem of a matrix is said to be complete if

- The matrix has k distinct eigenvalues, or
- For any eigenvalue that repeats r times, there are r distinct orthogonal eigenvectors.

B.3.2 Proof that the coupled system of ODE's arising from a periodic discretization in one space dimension really can be de-coupled

Theorem: *A banded periodic matrix $B_p(\ldots,a_{-2},a_{-1},a_0,a_1,a_2,\ldots)$ of size $i_{\max}$ (rows numbered from $i=1$) has a complete eigensystem. The right eigenvectors are of the form*

$$X_k = \left(1\; e^{I\phi_k}\; e^{2I\phi_k}\; \cdots\; e^{(i-1)I\phi_k}\; \cdots\; e^{(i_{max}-1)I\phi_k}\right)^T \tag{B.3}$$

where $\phi_k = 2\pi k/i_{max}$ for an integer $0 \le k < i_{max}$ and $I \equiv \sqrt{-1}$. In general, the ith element of the vector is $e^{(i-1)I\phi_k}$. The eigenvalues are

$$\lambda_k = \sum a_j e^{jI\phi_k} \tag{B.4}$$

Proof:

Note that Equations B.3 and B.4 define a set of vectors and values that are the right size to be a complete eigensystem. We merely need to show that each pair really is an eigenvector-eigenvalue pair; that is, that $B_pX_k = \lambda_kX_k$. Substituting the expressions above, we have, for a general vector element i:

$$\begin{aligned} B_p(\ldots,a_{-2},a_{-1},a_0,a_1,a_2,\ldots)X_k &= \cdots + a_{-2}\exp((i-3)I\phi_k) + a_{-1}\exp((i-2)I\phi_k) \\ &\quad + a_0\exp((i-1)I\phi_k) + a_1\exp((i)I\phi_k) + a_2\exp((i+1)I\phi_k)\cdots \\ &= e^{(i-1)I\phi_k}\left(\sum a_j\exp(jI\phi_k)\right) \\ &= \lambda_kX_k \end{aligned}$$

The wrap-around cases at the ends of the vector are also correct. For example, consider the first element of $B_p X_k$:

$$\begin{aligned}
\cdots a_{-2} e^{(i_{max}-2)I\phi_k} + a_{-1} e^{(i_{max}-1)I\phi_k} + a_0 + a_1 e^{I\phi_k} + a_2 e^{2I\phi_k} \cdots &= \\
\cdots a_{-2} \exp(i_{max} I\phi_k) \exp(-2I\phi_k) + a_{-1} \exp(i_{max} I\phi_k) \exp(-I\phi_k) & \\
+ a_0 + a_1 e^{I\phi_k} + a_2 e^{2I\phi_k} \cdots &= \\
\cdots a_{-2} e^{-2I\phi} + a_{-1} e^{-I\phi} + a_0 + a_1 e^{I\phi_k} + a_2 e^{2I\phi_k} \cdots &= \lambda_k X_k[1]
\end{aligned}$$

where the last transformation is possible because:

$$i_{max}\phi_k = i_{max}\frac{2\pi k}{i_{max}} = 2\pi k$$

and

$$e^{2\pi kI} = \cos 2\pi k + I \sin 2\pi k = 1$$

Q.E.D.

Appendix C

Solution of Tri-Diagonal Systems of Equations

C.1 The Thomas Algorithm

A system of i_{max} equations requires, in general, $O\left(i_{max}^3\right)$ operations to solve. However, if the matrix on the left-hand side of the equation is tri-diagonal in form, we can solve the system using the Thomas algorithm, which requires only $O\left(i_{max}\right)$ operations.

Consider a general tri-diagonal system of equations

$$\begin{bmatrix} b_1 & c_1 & & & & & \\ a_2 & b_2 & c_2 & & & & \\ & a_3 & b_3 & c_3 & & & \\ & & a_4 & b_4 & c_4 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & a_{i_{max}-1} & b_{i_{max}-1} & c_{i_{max}-1} \\ & & & & & a_{i_{max}} & b_{i_{max}} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{i_{max}-1} \\ x_{i_{max}} \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ \vdots \\ r_{i_{max}-1} \\ r_{i_{max}} \end{pmatrix}$$

The Thomas algorithm uses Gauss elimination and back substitution to solve these equations, taking advantage of the structure of the matrix to avoid unnecessary work. The following pseudo-code (in no particular language) shows how this is done. C and Fortran code for this algorithm is available on the course web site.

```
! First use linear combinations of rows to eliminate
```

```
! all the a's and scale each row to make all the b's
! equal to 1.
for i = 1, i_max - 1
  c_i ← c_i/b_i
  r_i ← r_i/b_i
  b_i ←
1                    ! This line has no effect
  b_{i+1} ← b_{i+1} - c_i a_{i+1}
  r_{i+1} ← r_{i+1} - r_i a_{i+1}
  a_{i+1} ←
0                  ! This line has no effect
end for
r_{i_max} ← r_{i_max}/b_{i_max}
b_{i_max} ←
1                    ! This line has no effect

! Now back-substitute, eliminating the c's.  After this
! pass the r's will have been replaced by the x's.
for i = i_max - 1, 1 by -1
  r_i ← r_i - r_{i+1} c_i
  c_i ←
0                    ! This line has no effect
end for
```

C.2 The Thomas Algorithm for Systems

The Thomas algorithm can be easily extended to systems of equations as well. The only subtle point here is that, instead of dividing by a matrix, we need to left-multiply by its inverse.

Pseudo-code for solving the system of $i_{\max}$ block equations of the form

$$A_i X_{i-1} + B_i X_i + C_i X_{i+1} = R_i$$

follows; as for the scalar case, C and Fortran code is available on the course web site.

```
! First use linear combinations of rows to eliminate
```

```
! all the A's and scale each row to make all the B's
! equal to I.
for i = 1, i_max - 1
  C_i ← B_i^{-1} C_i
  R_i ← B_i^{-1} R_i
  B_i ← I                    ! This line has no effect
  B_{i+1} ← B_{i+1} - A_{i+1} C_i
  R_{i+1} ← R_{i+1} - A_{i+1} R_i
  A_{i+1} ← 0                ! This line has no effect
end for
R_{i_max} ← B_{i_max}^{-1} R_{i_max}
B_{i_max} ← I                ! This line has no effect

! Now back-substitute, eliminating the C's.  After this
! pass the R's will have been replaced by the X's.
for i = i_max - 1, 1 by -1
  R_i ← R_i - C_i R_{i+1}
  C_i ← 0                    ! This line has no effect
end for
```

Appendix D

Programming Guidelines

There are two purposes for these guidelines. First, they will make your lives easier as you program and debug, both in this class and afterwards. Second, they will make my life easier as I mark your programming assignments and try to figure out where that one last bug is that you couldn't quite locate.

Indent loops, if-then-else, case statements, etc. So that you can read your code and locate at a glance the end of a loop or an if-then-else, having your code properly indented is invaluable. If you use emacs as your text editor, you're set: emacs will automatically indent code for you — just hit the TAB key; this is the way the samples were indented. If not, use at least two spaces of extra indentation for each level of loop or conditional nesting.

Comment your code Commenting is essential if you're going to ever figure out again what that complicated arithmetic expression is supposed to calculate, and so that you won't change that line of code that is correct but counter-intuitive. You needn't go overboard. Provide a comment that describes the purpose of each subprogram and then add comments within subprograms to describe the major things that it does.

Write modular code In the long run, you'll save yourself a tremendous amount of programming time and effort if you write modular code, because you'll be able to directly re-use old pieces of code. For example, the exact same tri-diagonal equation

solver can be used in this course to solve Poisson's equation, the heat equation, and the energy equation.

Another advantage to writing modular code is that it helps you narrow down the location of errors in the program more easily.

Finally, the thought required to write modular code is likely to help you organize your understanding of the physics and/or numerics of a problem, which will help you write correct code.

Declare and type all your variables Explicitly declare your variables and their type (real, float, int, double precision, etc). This is a practical piece of advice designed to reduce your debugging time. If you accidentally type `tume` instead of `time` as a variable name, you would like the compiler to tell you this instead of having to find out the hard way. C compilers will do this automatically. In Fortran, either use an appropriate compiler flag (often `-u`) or add `implicit none` to the beginning of each subprogram.

Use every available compiler warning flag Another piece of practical advice: ask your compiler to be very picky in providing warnings about your program. Most compilers will happily tell you that all sorts of legal things are questionable (and possibly incorrect). It's much easier to find these errors at compile time. After years of following this advice, I still occasionally get a useful and unexpected warning from a compiler that I haven't run my mesh generation code through in a while. To find out how to do this, check your compiler's manual, either on paper or electronically.

D.1 Sample Program in C

```
#include <assert.h>
#include <stdio.h>
#include <unistd.h>
#include <math.h>

#define NMAX 256
double dTime = 0;
```

```
/* Compute the exact solution to the problem as a function of X and T.
static double dExact(const double dX, const double dTime);

/* Print out the computed solution, the exact solution, and the error i
   the computed solution. */
static void vPrintSoln(const double adSoln[NMAX], const int iN)
{
  double dDX = 1./iN;
  int i;
  for (i = 0; i <= iN; i++) {
    double dX = i * dDX;
    printf("%5d %10.6f %12.8f %12.8f %12.8g\n", i,
           dX, adSoln[i], dExact(dX, dTime),
           dExact(dX, dTime) - adSoln[i]);
  }
}

/* Compute the flux integral for all control volumes. */
static void vComputeResidual(const double adSoln[NMAX],
                             const int iN, double adResid[NMAX])
{
  int i;
  for (i = 1; i < NMAX; i++) {
    adResid[i] = 0;
  }

  /* Interior scheme */
  for (i = 1; i < NMAX-1; i++) {
    double dFlux = ...;
    adResid[i] += dFlux;
    adResid[i-1] -= dFlux;
  }

  /* Boundary conditions */
  /* At left boundary */
  adResid[1] += ...;
  /* At right boundary */
  adResid[NMAX-1] -= ...;
```

```
}

/* Performs a simple first-order explicit time march. */
static void vTimeMarch(double adSoln[NMAX], const int iN,
                       const double dDT)
{
  double adResid[NMAX];
  int i;

  /* First compute the flux integral */
  vComputeResidual(adSoln, iN, adResid);

  /* Advance the solution in time */
  for (i = 1; i < iN; i++)
    adSoln[i] += adResid[i] * dDT;

  /* Update the global simulation time. */
  dTime += dDT;
}

/* Set up initial condition */
static void vInit(double adSoln[], const int iN);

int main(int iNArgs, char *apcArgs[])
{
  int iN = 40, iNTimeSteps;

  double dTMax = 1., dDT, dCFL;
  double adSoln[NMAX];

  vInit(adSoln, iN);

  iNTimeSteps = 20;

  dDT = dTMax / iNTimeSteps;

  fprintf(stderr, "Running %d time steps at dt = %6.4f.\n",
          iNTimeSteps, dDT);
```

```
  fprintf(stderr, "Total time is %6.4f\n", dTMax);
  fprintf(stderr, "Spatial resolution is %6.4f (%d points)\n\n", 1./iN,

  {
    int iStep;
    for (iStep = 0; iStep < iNTimeSteps; iStep++)
      vTimeMarch(adSoln, iN, dDT);
  }

  vPrintSoln(adSoln, iN);
  exit(0);
}
```

D.2 Sample Program in Fortran

```
      program main
      implicit none
      integer NMAX
      parameter (NMAX = 256)
      double precision Time
      integer iN, iNTimeSteps, iStep
      double precision TMax, DT, Soln(NMAX)

      iN = 40
      TMax = 1.

      call Init(Soln, iN)

      iNTimeSteps = 20

      DT = TMax / iNTimeSteps

      write(6,10) iNTimeSteps, DT
      write(6,20) TMax
      write(6,30) 1./iN, iN
```

```
 10   format("Running",I6," time steps at dt =",F7.4)
 20   format("Total time is ",F6.4)
 30   format("Spatial resolution is",F7.4," (",I5," points)")

      do iStep = 1, iNTimeSteps
         call TimeMarch(Soln, iN, Time, DT)
      enddo

      call PrintSoln(Soln, Time, iN)
      stop
      end

C     Compute the exact solution to the problem as a function
      double precision function Exact(X, Time)
      implicit none
      integer NMAX
      parameter (NMAX = 256)
      double precision X, Time
      ...
      Exact = ...
      return
      end

C     Print out the computed solution, the exact solution, an
C     the computed solution.
      subroutine PrintSoln(Soln, Time, iN)
      implicit none
      integer NMAX
      parameter (NMAX = 256)
      double precision Soln(*), DX, X, Exact, Time
      integer iN, i

      DX = 1./iN
      do i = 1, iN
         X = i * DX
         write(6,10) X, Soln(i), Exact(X, Time),
     $        Exact(X, Time) - Soln(i)
```

```
      enddo
 10   format(I5,F10.6,2F12.8,G12.8)
      return
      end

C     Compute the flux integral for all control volumes.
      subroutine ComputeResidual(Soln, iN, Resid)
      implicit none
      integer NMAX
      parameter (NMAX = 256)
      double precision Soln(*), Resid(*), Flux
      integer iN, i

      do i = 1, NMAX
         Resid(i) = 0
      enddo

C     Interior scheme
      do i = 1, NMAX-1
         Flux = ...
         Resid(i)   = Resid(i)   + Flux
         Resid(i-1) = Resid(i-1) - Flux
      enddo

C     Boundary conditions
C     At left boundary
      Resid(1) = Resid(1) + ...
C     At right boundary
      Resid(NMAX-1) = Resid(NMAX-1) - ...

      return
      end

C     Performs a simple first-order explicit time march.
      subroutine TimeMarch(Soln, iN, Time, DT)
      implicit none
      integer NMAX
      parameter (NMAX = 256)
```

```
      double precision Soln(*), DT, Resid(NMAX), Time
      integer iN, i

C     First compute the flux integral
      call ComputeResidual(Soln, iN, Resid)

C     Advance the solution in time
      do i = 1, iN
         Soln(i) = Soln(i) + Resid(i) * DT
      enddo

C     Update the global simulation time.
      Time = Time + DT
      return
      end

C     Set up initial condition
      subroutine Init(Soln, iN)
      implicit none
      integer NMAX
      parameter (NMAX = 256)
      double precision Soln(*)
      integer iN
      ...
      return
      end
```

D.3 Painless Array Manipulation

The most common data structure (in fact, often the only data structure) in CFD programs is the array. Therefore it's important that you be able to declare arrays properly and to pass them as arguments to subroutines. The details of this differ in

C and Fortran[1], but the idea is the same.

D.3.1 Array Declaration

Suppose that you need to store a solution on a finite-volume mesh that has 24 volumes in the i-direction and 36 volumes in the j-direction. This data should be stored in an array of size 26×38, to allow room for ghost cell data. I personally like to number cells so that the ghost cells start at 0, so I would declare this array in C as:

```
double solution[26][38];
```

or more likely:

```
#define ISIZE 24
#define JSIZE 36
...
double solution[ISIZE+2][JSIZE+2];
```

I prefer the second choice because the symbolic constants `ISIZE` and `JSIZE` can be re-used in other places where these values are needed, and the mesh size can be changed easily by redefining `ISIZE` and `JSIZE`.

In Fortran, I would write

```
double precision solution(0:25,0:37)
```

Fortran 77 has no portable way to define symbolic global constants, but many compilers allow you to use preprocessor directives as I've done in the C example. Another non-standard solution is to declare sizes as `PARAM`'s in an include file, and include that file into every routine that needs it. This is also possibly non-portable, but most (all?) Fortran compilers allow includes now.

[1] I'll use Fortran 77 examples throughout, because Fortran 90 compilers have still not taken over completely and because I have never used Fortran 90 personally.

D.3.2 Passing Arrays as Arguments to Subroutines

Once you have an array that contains your solution, you'll need to pass it to a variety of subroutines for initialization, flux calculation, solution update, output, etc. This is a delicate subject in C, where you must know the size of the array in advance;[2] this is another place where `#define`-ing array sizes makes life simpler. In C:

```
void Resid(const double solution[ISIZE+2][JSIZE+2],
           const int IMax, const int JMax,
           double residual[ISIZE][JSIZE])
```

Notice that the `residual` (which is calculated only for the interior cells) has a different size than the solution in this example.

In Fortran, the same effect is achieved differently.

```
subroutine Resid(solution, IMax, JMax, ISize, JSize,
                 residual)
integer IMax, JMax, ISize, JSize
double precision solution(0:ISize+1,0:JSize+1)
double precision residual(ISize,JSize)
```

Note that Fortran does allow the size of an array passed to a subroutine to be determined at run time (as an argument to the subroutine).

D.3.3 Passing Array Slices as Arguments

Suppose that you want to send only part of your array (a single line of data, say) as an argument to a subroutine. You will see an application of this when we get to approximate factorization (see Section 8.4). C and Fortran each allow you to pass an array slice, but it's a different slice, depending on the language. In C, you would write:

[2] Technically, you can avoid this by using an array of pointers to `double` instead of a two-dimensional array of `double`. However, this requires that you allocate the memory block that each of those pointers points to, which is annoying and error-prone. If you understand this paragraph and want to try it that way, go ahead and experiment.

```
FuncNeedingConstantISlice(solution[i]);
```

where `FuncNeedingConstantISlice` takes a one-dimensional array of doubles as an argument.

In Fortran, on the other hand, you would pass an array slice like this:

```
call FuncNeedingConstantJSlice(solution(1,j));
```

The difference between how the two languages handle passing array slices arises from the order in which elements of multi-dimensional arrays are stored. To pass an array slice in the "non-native" direction, you must copy data to a temporary array and pass that temporary.[3]

D.3.4 Higher-dimensional Arrays

Let's suppose that you want to store three 3×3 matrices for every cell in each column of your mesh.[4] The handling of such an array is just more of the same that we've done before. In C, you might write:

```
double LHS[JSIZE][3][3][3];
```

This order makes it possible to refer to a 3×3 block by writing `LHS[j][k]` or to a row in the block tri-diagonal matrix as `LHS[j]`.

In Fortran, one would write

```
double precision LHS(3,3,3,JSIZE)
```

and refer to a 3×3 block by writing `LHS(1,1,k,j)` or to a row in the block tri-diagonal matrix as `LHS(j)`.

This reversal in the order of indices is only likely to be confusing if you switch back and forth between languages often.

[3] Fortran 90 introduced a new interface for array slicing into Fortran that does this copying for you.

[4] You might want to do this to solve a block tri-diagonal system of equations when simulating a coupled system of PDE's.

D.4 Most Popular CFD Programming Errors

Mis-sized arrays. A very common error in CFD programming is to size arrays slightly too small. The most common form of this error is to forget to allow space for ghost cells. Remember: 100 cells in the interior means you need 102 cells total.

Off-by-one loop errors. Starting too late or ending too soon will leave some data unchanged. Starting too early or ending too late may cause you to overwrite data other than the array you intended to change.

Array size mismatches. One of the most baffling bugs I've seen in a CFD code was caused by having an array declared as

```
double name[332][3];
```

and passed as an argument to a routine that wanted

```
double name[322][3];
```

The size difference in arrays matters. This is a really good reason to use symbolic constants (either `#define`'s in C or the equivalent in Fortran).

Not initializing variables. Don't count on an uninitialized variable having a sane value, because it may not. Even if it does, it may still not be the right value. Most compilers will catch the more obvious cases of this, but not all. For example, if you accumulate a flux integral by adding each new piece of the integral to the total you already have, you need to be sure to initialize the totals to zero each time before you start.

Sign errors. These are extremely common. I often find myself thinking, "But that sign *has* to be right!" I've learned over the years that it's to my advantage to change signs that are in question and run the code again. If the code's behavior improves, then the sign was wrong, and I have to re-calibrate my brain so that I'll understand why. This approach is generally a lot faster, at least for me, than re-verifying analytically what the sign should have been, especially since I've been known to derive the same wrong answer more than once in a row.

Appendix E

References

Mathematics

For Taylor series, a good intro calculus book.

For Gauss's theorem, a good multi-variable calculus book.

For help finding exact comparison solutions to PDE's, *Advanced Engineering Mathematics* by Wylie and Barrett is a pretty good choice. I'm sure there are others that are equally good.

Fluid Dynamics

Frank White's textbooks are both very good. *Fluid Mechanics* at the undergraduate level and *Viscous Fluid Flow* at the graduate level.

Batchelor's *Introduction to Fluid Mechanics* is also an excellent graduate-level text.

Computational Fluid Dynamics

Numerical Computation of Internal and External Flows, by Charles Hirsch. In two volumes. A great reference for compressible flow methods. Doesn't cover a lot about incompressible flow. I'd probably pick this as my sole reference if I had to pick just one.

Fundamentals of Computational Fluid Dynamics, by Lomax, Pulliam and Zingg. Second Edition, 2003. Great for modern analysis (better than Anderson, Tannehill, and Pletcher, as far as I'm concerned). This is the one I'd add to Hirsch if I were allowed to pick two.

Computational Methods for Fluid Dynamics, by Joel Ferziger. I've not actually taken a close look at this book, but it's highly thought of.

Numerical Heat Transfer and Fluid Flow, by S. V. Patankar. I disagree with Patankar's choices in discretization schemes (even leaving aside issues of finite difference versus finite volume methods). This doesn't mean it's a bad book, or even that his choices are incorrect.

Computational Fluid Mechanics and Heat Transfer, by Anderson, Tannehill, and Pletcher. Good coverage of the basics of finite difference methods, including analysis of accuracy and stability. Other parts of the book are a bit dated.