

Mech 510  
Computational Methods in Transport  
Phenomena I

Carl Ollivier-Gooch

UBC Department of Mechanical Engineering



# Contents

<b>1</b>	<b>Intro to CFD</b>	<b>1</b>
1.1	Modeling . . . . .	1
1.2	Discretization . . . . .	4
1.3	Accuracy and Stability . . . . .	5
1.4	Validation . . . . .	6
1.5	Efficiency . . . . .	8
1.6	Convergence . . . . .	8
<b>2</b>	<b>Modeling Based on the Navier-Stokes Equations</b>	<b>9</b>
2.1	Non-dimensionalization of the Navier-Stokes equations . . . . .	9
2.2	Derivation of model problems . . . . .	13
<b>3</b>	<b>Space Discretization of PDE's</b>	<b>15</b>
3.1	Overview . . . . .	15
3.2	Transformation of a PDE into Control Volume Form . . . . .	22
3.3	Second-order Accurate Flux for the Poisson Equation . . . . .	23
3.4	Flux Integrals . . . . .	26
3.5	Problems . . . . .	27
<b>4</b>	<b>Accuracy Assessment for Numerical Solutions</b>	<b>29</b>
4.1	If an exact solution is available . . . . .	29
4.2	If an exact solution is <i>not</i> available . . . . .	30
4.3	Problems . . . . .	31

<b>5</b>	<b>Time Accuracy and Stability Analysis for Ordinary Differential Equations</b>	<b>33</b>
5.1	From PDE to Coupled ODE's . . . . .	33
5.2	Analysis of Time March Schemes for ODE's . . . . .	35
5.3	Caveats . . . . .	37
5.4	Examples . . . . .	37
5.5	Stability Analysis for Fully-Discrete Systems . . . . .	43
5.6	Examples . . . . .	44
5.7	Problems . . . . .	45
<b>6</b>	<b>Systems of PDE's</b>	<b>47</b>
6.1	Computation of Flux and Source Jacobians . . . . .	49
6.2	Problems . . . . .	52
<b>7</b>	<b>Practical Aspects of Solving Poisson's Equation</b>	<b>55</b>
7.1	Solving the Discrete Poisson Equation . . . . .	55
7.2	Boundary Conditions for the Laplacian . . . . .	60
<b>8</b>	<b>The Wave Equation</b>	<b>65</b>
8.1	Boundary Conditions for the Wave Equation . . . . .	66
8.2	Basic Results for the Wave Equation . . . . .	67
8.3	Advanced Schemes for the Wave Equation . . . . .	70
<b>9</b>	<b>The Incompressible Energy Equation</b>	<b>83</b>
9.1	Simple Discretization of the Incompressible Energy Equation . . . . .	85
9.2	Time Discretization of the Energy Equation . . . . .	86
9.3	Approximate Factorization . . . . .	90
9.4	Boundary Conditions . . . . .	91
9.5	Outline of Navier-Stokes Code . . . . .	95

<i>CONTENTS</i>	iii
<b>A Glossary</b>	<b>97</b>
<b>B Some Mathematical Concepts Useful for CFD</b>	<b>101</b>
B.1 Classification of PDE's . . . . .	101
B.2 Taylor Series Expansions . . . . .	101
B.3 Eigenvalues, Eigenvectors, and All That . . . . .	102
<b>C Solution of Tri-Diagonal Systems of Equations</b>	<b>105</b>
C.1 The Thomas Algorithm . . . . .	105
C.2 The Thomas Algorithm for Systems . . . . .	106
<b>D Programming Guidelines</b>	<b>109</b>
D.1 Sample Program in C . . . . .	110
D.2 Sample Program in Fortran . . . . .	113
D.3 Painless Array Manipulation . . . . .	116
D.4 Most Popular CFD Programming Errors . . . . .	120
<b>E Validating CFD Programs</b>	<b>121</b>
E.1 Modeling . . . . .	122
E.2 Validation . . . . .	123
<b>F References</b>	<b>125</b>



# List of Figures

1.1	Schematic representation of finite difference approximation to a continuous solution. . . . .	4
1.2	Schematic representation of finite volume approximation to a continuous solution. . . . .	5
1.3	Schematic representation of finite element approximation to a continuous solution. . . . .	5
3.1	Flux integration around a finite volume. . . . .	26
7.1	Comparison of convergence rates for point iterative schemes applied to Laplace's equation. . . . .	59
7.2	Comparison of convergence rates for point and line Gauss-Seidel iterative schemes applied to Laplace's equation. . . . .	60
7.3	Finite volume with homogeneous Neumann boundary condition imposed along once side. . . . .	61
7.4	Boundary cell showing Dirichlet boundary condition and ghost cell. . . . .	62
8.1	First-order time advance for the wave equation with several space discretizations . . . . .	68
8.2	Second-order time advance for the wave equation with several space discretizations . . . . .	69
8.3	Second-order time advance for the wave equation propagating a square wave. . . . .	69
8.4	Effect of mesh refinement on square wave propagation using the second-order upwind scheme. . . . .	70

8.5	Example control-volume averaged solution . . . . .	71
8.6	Legal values of $\psi(r)$ for TVD schemes . . . . .	72
8.7	Three TVD limiters . . . . .	74
8.8	Upwind TVD schemes (square wave) . . . . .	79
8.9	ENO and FCT schemes (square wave) . . . . .	80
8.10	Propagation of a smooth solution (sine wave) . . . . .	81



# Chapter 1

## Intro to CFD

### Strengths of Computational/Experimental/Analytic Fluid Dynamics

#### 1.1 Modeling

Consider the flow inside a commercial tire incinerator (picture shown in class). Ground-up tires are dumped in and burned. Gases pass by several heat exchangers to boil and superheat water for power generation. Because of the combustion conditions, there is a significant amount of nitrogen oxides ( $NO_x$ ) in the flue gases, which is environmentally unacceptable. A company in Illinois (Nalco Fueltech) makes a living by selling  $NO_x$  reduction systems for incinerators like this one. They need an accurate CFD model that can be easily applied to a variety of incinerators rather quickly so that they can design  $NO_x$  reduction systems. In particular, they must predict temperature, velocity, and  $NO_x$  concentration both with and without their emission reduction system for several operating conditions. They must do this quickly (a couple of weeks at most).

Key features of the physics: Governing equations that include most of these effects:

Global continuity:

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

Momentum:

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

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*)
- Neglect radiative heat transfer

This reduces the mathematical description of the problem to:

Global continuity:

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

Momentum:

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

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.

*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.

## 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 a topic that we will discuss in Mech 511.

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)

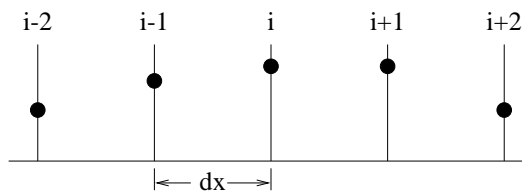


Figure 1.1: Schematic representation of finite difference approximation to a continuous solution.

- The original approach for CFD.
- Easy to get high-order discretizations (use high-order finite differences).
- Doesn't 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.
- Applicable to any mesh topology w/ appropriate control volumes.

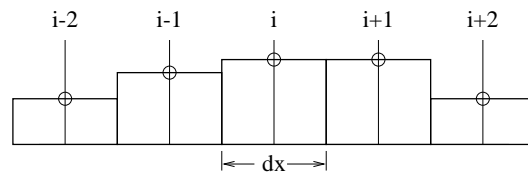


Figure 1.2: Schematic representation of finite volume approximation to a continuous solution.

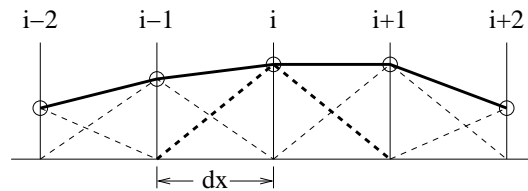


Figure 1.3: Schematic representation of finite element approximation to a continuous solution.

- Not too hard to get high-order discretizations (locally construct high-order polynomial representation of 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 is difficult, but possible.

## 1.3 Accuracy and Stability

The finite representation of the PDE is not exact; some *discretization 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. We also find out how the discretization error will change as we add mesh points.

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 Validation

Then 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. *Validation* tells us whether the solutions we get for a series of simple test cases are correct. *Debugging* is the process of identifying *why* a program failed a test case and fixing it. A validation 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.

- 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.
- 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-validated 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.

- Table 1.1 gives a partial listing of test problems worth considering for the tire incinerator problem.

Case	Physics	Change in physics
1	Inviscid terms only, no heat addition, no flow initial condition in a closed rectangular box (analytic solution)	
2	Inviscid terms only, no heat addition, no flow initial condition with wall, inflow, and outflow boundary conditions (analytic solution)	Different boundary conditions
3	Inviscid terms only, no heat addition, uniform flow in a straight rectangular duct (analytic solution)	Non-zero velocity
4	Inviscid terms only, no heat addition, accelerating flow in a straight rectangular duct (analytic solution)	Velocity increasing to steady-state
n-3	Turbulent flow without heat addition in a straight duct (experimental data)	Turbulence
n-2	Turbulent flow with heat addition in a straight duct (experimental data)	Heat addition
n-1	Turbulent flow without heat addition in a duct with abrupt turns (experimental data)	Flow around bends
n	Turbulent flow with heat addition in tire incinerator	Combines n-2 and n-1

Table 1.1: Partial validation plan for the tire incinerator problem.

Technically, the listing of test cases for the tire incinerator problem mixes validation cases and verification cases. The difference between these two categories is that

*verification* ensures that your program correctly implements the physics that you intended it to, while *validation* demonstrates that your physical model comes close enough to reality for the problem of interest.

## 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
- Simplify the discretization
- Improve the technique we use to solve the discretized equations
- 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

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

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \quad (2.1)$$

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

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

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

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  $u_{\text{ref}}$ , pressure by  $\rho u_{\text{ref}}^2$ , and temperature by  $T_{\text{ref}}$ . Basically, we just assume that we can find some appropriate reference values  $L$ ,  $u_{\text{ref}}$  and  $T_{\text{ref}}$  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  $u_{\text{ref}}$ . 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:

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

or

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

or

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

## 2.1. NON-DIMENSIONALIZATION OF THE NAVIER-STOKES EQUATIONS 11

Substituting into the x-momentum equation:

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

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

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

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

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

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

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

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

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

What are the dimensionless parameters here?

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

and

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

Summarizing the non-dimensional equations,

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

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

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

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

Note the extreme similarity in form between Equations 2.1–2.4 on the one hand and Equations 2.5–2.8. 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:

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

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.
- 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.

### 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

Suppose we have a general conservation law (with source term) of the form<sup>1</sup>

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

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:

$$\frac{\partial T}{\partial t} + \frac{\partial uT}{\partial x} = \alpha \frac{\partial^2 T}{\partial x^2}$$

---

<sup>1</sup>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.

where  $u$  and  $\alpha$  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  $O(\Delta x^2)$ ; 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  $O(\Delta x^2)$ . 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 happens to be 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

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

where the  $b_i$  are *basis functions*, and the  $T_i$  are the nodal solution values (which 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:

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

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 volume 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  $w_i$ :

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

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

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

Here we've used integration by parts, and note that the first term on the left 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:

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

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

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:

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

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:

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

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

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

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

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:

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

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:

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

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  $\bar{T}_{N+1}$ ), 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.

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

### 3.3. SECOND-ORDER ACCURATE FLUX FOR THE POISSON EQUATION 23

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.<sup>2</sup>

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.

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

This equation states that the average value  $\bar{U}$  of the solution in the control volume changes at a rate determined by the 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:

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

---

<sup>2</sup>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  $O(\Delta x^2)$  by taking its value as  $\bar{S} \approx S(\bar{U})$ .

Integrating over control volumes, we have

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

$$\int_{CV} \nabla \cdot \left( \frac{\partial T}{\partial x} \hat{i} + \frac{\partial T}{\partial y} \hat{j} \right) dA = \bar{S}A \quad (3.7)$$

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

The last transformation uses Gauss's theorem. So the flux in Poisson's equation is  $\left( \frac{\partial T}{\partial x} \hat{i} + \frac{\partial T}{\partial y} \hat{j} \right)^T$ . 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

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

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:

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

We can use this to calculate the flux we need, because we know that, for a sufficiently fine mesh, we will get the correct derivative. While it's very comforting to know this, it would be even better if we knew how quickly the error in the approximation approaches zero.

To determine this, expand each term on the right-hand side of Equation 3.9 in a Taylor series expansion about  $x_0$ :

$$\begin{aligned} T(x_0 + \varepsilon) &= T(x_0) + \varepsilon \left. \frac{dT}{dx} \right|_{x_0} + \frac{\varepsilon^2}{2} \left. \frac{d^2T}{dx^2} \right|_{x_0} + \frac{\varepsilon^3}{6} \left. \frac{d^3T}{dx^3} \right|_{x_0} + \dots \\ T(x_0 - \varepsilon) &= T(x_0) - \varepsilon \left. \frac{dT}{dx} \right|_{x_0} + \frac{\varepsilon^2}{2} \left. \frac{d^2T}{dx^2} \right|_{x_0} - \frac{\varepsilon^3}{6} \left. \frac{d^3T}{dx^3} \right|_{x_0} + \dots \end{aligned}$$

Combining these,

$$\frac{T(x_0 + \varepsilon) - T(x_0 - \varepsilon)}{2\varepsilon} = \left. \frac{dT}{dx} \right|_{x_0} + \frac{\varepsilon^2}{6} \left. \frac{d^3T}{dx^3} \right|_{x_0} + O(\varepsilon^4) \quad (3.11)$$



### 3.3. SECOND-ORDER ACCURATE FLUX FOR THE POISSON EQUATION 25

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.

	$T(x_0)$	$\frac{\partial T}{\partial x} \Big _{x_0}$	$\frac{\partial^2 T}{\partial x^2} \Big _{x_0}$	$\frac{\partial^3 T}{\partial x^3} \Big _{x_0}$
$\frac{T(x_0+\varepsilon)}{2\varepsilon}$	$\frac{1}{2\varepsilon}$	$\frac{1}{2}$	$\frac{\varepsilon}{4}$	$\frac{\varepsilon^2}{12}$
$-\frac{T(x_0-\varepsilon)}{2\varepsilon}$	$-\frac{1}{2\varepsilon}$	$\frac{1}{2}$	$-\frac{\varepsilon}{4}$	$\frac{\varepsilon^2}{12}$
$\frac{T(x_0+\varepsilon)-T(x_0-\varepsilon)}{2\varepsilon}$	0	1	0	$\frac{\varepsilon^2}{6}$

The *truncation error* in a difference approximation  $D$  of a differential operator  $\mathbf{D}$  is defined to be  $D - \mathbf{D}$ .<sup>3</sup> An approximation is said to be  $k^{\text{th}}$ -order accurate if and only if the leading-order term in the truncation error is  $O(\varepsilon^k)$ .

For our example, the truncation error is  $\frac{\varepsilon^2}{6} \frac{\partial^3 T}{\partial x^3} \Big|_{x_0} + O(\varepsilon^4)$ . This approximation is therefore second-order accurate, and the error in the approximation will fall by a factor of four each time  $\varepsilon$  is reduced by a factor of two.

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

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

and

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

But can we really justify the use of  $\bar{T}_{i+1,j}$  rather than a pointwise value of  $T$  evaluated at the center of control volume  $(i+1, j)$ ? Yes, in fact we can, so long as we're only looking at first- or second-order accuracy; it's not hard to show that the difference between the control volume average and the local value at the control volume centroid for a smooth function and a structured mesh is second order.

<sup>3</sup>You may also see this definition with the sign reversed; the difference is largely philosophical.

### 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

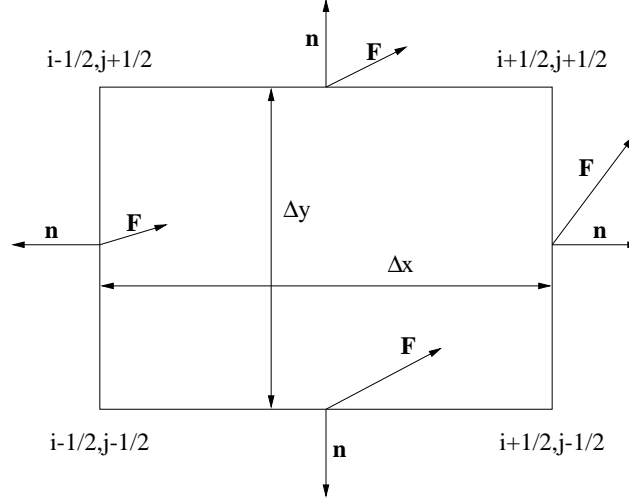


Figure 3.1: Flux integration around a finite volume.

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

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

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

$$\oint_{\partial CV} \vec{F} \cdot \vec{n} dl = (\bar{T}_{i+1,j} - 2\bar{T}_{i,j} + \bar{T}_{i-1,j}) \frac{\Delta y}{\Delta x} + (\bar{T}_{i,j+1} - 2\bar{T}_{i,j} + \bar{T}_{i,j-1}) \frac{\Delta x}{\Delta y}$$

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.

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

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  $O(\Delta x^2)$ . (Hint: expand  $T$  in a Taylor series about  $x = x_i$ .)
2. Show that

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

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:  $\bar{T}_{i+2}$ ,  $\bar{T}_{i+1}$ ,  $\bar{T}_i$ , and  $\bar{T}_{i-1}$ . Find the most accurate possible approximation to the  $\frac{\partial T}{\partial x}$  at  $i + \frac{1}{2}$  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  $i + \frac{1}{2}$ :  $\bar{T}_i$  and  $\bar{T}_{i-1}$ . Find the most accurate possible approximation to the flux and determine the leading-order truncation error term.

# Chapter 4

## Accuracy Assessment for Numerical Solutions

### 4.1 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  $\Delta x$ . The error  $E_{i,j}$  in the numerical solution is:

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

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:

$$\|E_{i,j}\|_1 = \frac{\sum_i \sum_j |E_{i,j}|}{i_{max} j_{max}} \quad (4.1)$$

$$\|E_{i,j}\|_2 = \sqrt{\frac{\sum_i \sum_j E_{i,j}^2}{i_{max} j_{max}}} \quad (4.2)$$

$$\|E_{i,j}\|_\infty = \max_{i,j} |E_{i,j}| \quad (4.3)$$

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  $\Delta x$ , 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. 4.1) and  $L_2$  (Equation. 4.2) often converge a half or full order faster than the  $L_\infty$  norm (Equation. 4.3), which is a local measure. That is, the  $L_\infty$  norm can converge as  $O(\Delta x)$  because of a local error at a point, while the  $L_2$  norm will converge as  $O(\Delta x^{3/2})$  and the  $L_1$  norm will converge as  $O(\Delta x^2)$ . While this is not *always* true, it does happen sometimes.

## 4.2 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:

$$\begin{aligned} u|_{M_1} &= u_e + C\Delta x^k \\ u|_{M_2} &= u_e + C\left(\frac{\Delta x}{2}\right)^k \\ u|_{M_3} &= u_e + C\left(\frac{\Delta x}{4}\right)^k \end{aligned}$$

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.

## 4.3 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
$10 \times 10$	$4.68 \cdot 10^{-2}$	—
$20 \times 20$	$9.08 \cdot 10^{-3}$	5.15
$40 \times 40$	$2.13 \cdot 10^{-3}$	4.26
$80 \times 80$	$5.32 \cdot 10^{-4}$	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	$L_2$
$20 \times 20$	$40 \times 40$	$1.25 \cdot 10^{-3}$
$40 \times 40$	$80 \times 80$	$1.78 \cdot 10^{-4}$
$80 \times 80$	$160 \times 160$	$2.55 \cdot 10^{-5}$

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	$L_1$	$L_2$	$L_\infty$
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 5

## 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.

### 5.1 From PDE to Coupled ODE's

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

$$\frac{\partial T}{\partial t} \equiv \frac{dT_i}{dt} = a_{-2}T_{i-2} + a_{-1}T_{i-1} + a_0T_i + a_1T_{i+1} + a_2T_{i+2}$$

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*. 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

$$\frac{d\vec{T}}{dt} = B_p(a_{-2}, a_{-1}, a_0, a_1, a_2) \vec{T} \quad (5.1)$$

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 5.1 can always be derived. 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 5.1:

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

where  $\vec{w} \equiv X^{-1} \vec{u}$  is a new set of unknowns and  $\Lambda$  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 5.1.

**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.<sup>1</sup> 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  $G$  for any eigenvalue in the complex plane. This information is independent of the spatial scheme that produced the eigenvalue; the time advance analysis needs no information about the spatial discretization, not even the differential operator.

## 5.2 Analysis of Time March Schemes for ODE's

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

$$\frac{dw}{dt} = \lambda w \tag{5.2}$$

---

<sup>1</sup>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.

The exact solution of this equation is:

$$w(t) = Ae^{\lambda t} \quad (5.3)$$

Consequently,  $w$  grows exponentially in time when  $\Re(\lambda) > 0$  ( $\equiv$ 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*  $\sigma$  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 5.2 is given by

$$\sigma_{\text{exact}} \equiv \frac{w(t + \Delta t)}{w(t)} \equiv \frac{w^{n+1}}{w^n} = e^{\lambda \Delta t} = 1 + \lambda \Delta t + \frac{(\lambda \Delta t)^2}{2} + \frac{(\lambda \Delta t)^3}{6} + \dots \quad (5.4)$$

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 5.2. We'll examine a number of alternatives. In each case, we'll replace  $\frac{dw}{dt}$  and  $\lambda w$  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 5.4 are matched by the discrete scheme: the order of accuracy is equal to the highest-order term that *matches* the exact amplification factor.<sup>2</sup> A time advance scheme is said to be *stable* for all complex eigenvalues  $\lambda$  for which the magnitude of the complex amplification factor  $|\sigma| \leq 1$ .

We can combine the analysis results for a spatial scheme (the eigenvalues  $\lambda$  as a function of the spatial step  $\Delta x$ ) with the results for a time scheme (the amplification factor  $\sigma$  as a function of the eigenvalues  $\lambda$  and time step  $\Delta t$ ) to determine the stability properties of a particular space/time discretization (the amplification factor  $\sigma$  as a function of  $\Delta x$  and  $\Delta t$ ). 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 5.5 for more information about this.

---

<sup>2</sup>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.

## 5.3 Caveats

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.

**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.

## 5.4 Examples

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

### 5.4.1 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

$$\frac{dT}{dt}_i = \alpha \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2} \quad (5.5)$$

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

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

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

### 5.4.2 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:

$$\begin{aligned}\frac{\partial \bar{T}_i}{\partial t} &= -u \frac{\bar{T}_i - \bar{T}_{i-1}}{\Delta x} \\ &= B_p \left( \frac{u}{\Delta x}, -\frac{u}{\Delta x}, 0 \right)\end{aligned}$$

Then the eigenvalues can be written down immediately:

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

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

In the following series of examples, each time advance scheme is analyzed by using the model ODE defined in Equation 5.2 to determine both the accuracy and stability of the scheme.

### 5.4.3 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 5.2), this gives:

$$\frac{w^{n+1} - w^n}{\Delta t} = \lambda w^n \tag{5.6}$$

or

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

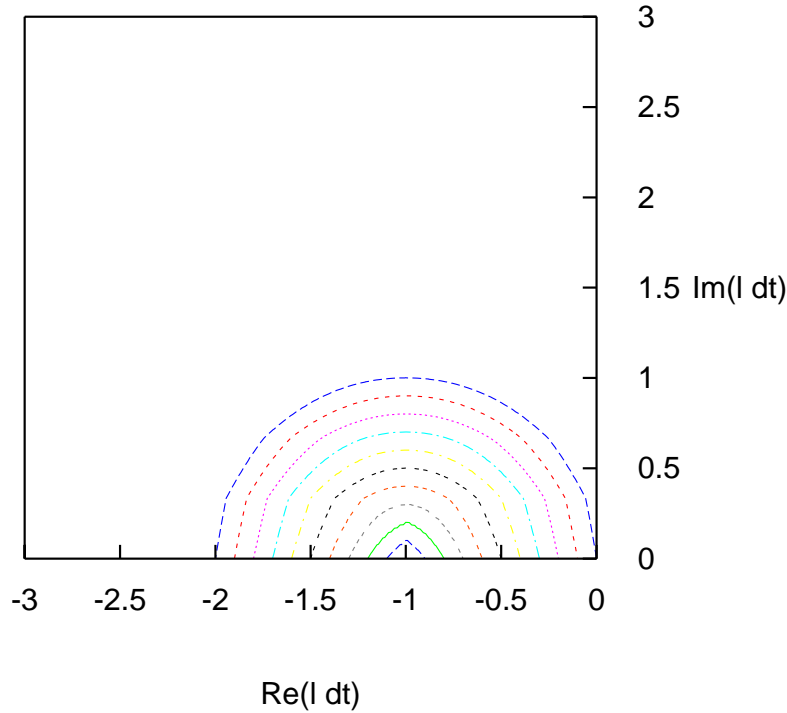
which implies that

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

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.

#### 5.4.4 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 5.2), this gives:

$$\frac{w^{n+1} - w^n}{\Delta t} = \lambda w^{n+1} \quad (5.7)$$

or

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

which implies that

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

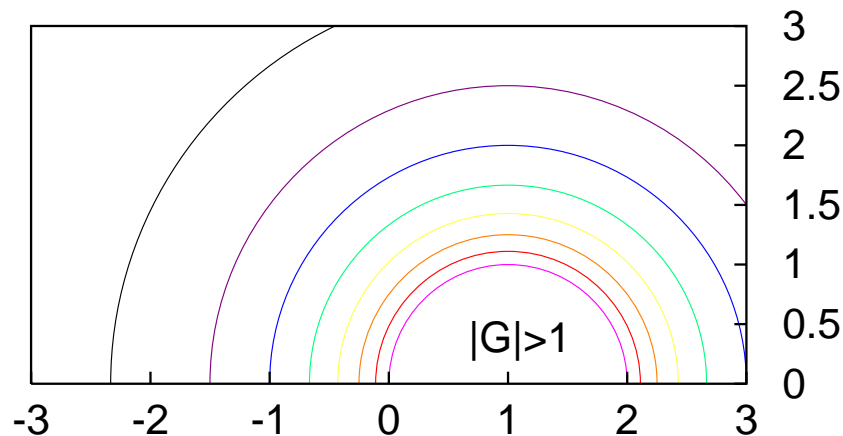
To determine the order of accuracy of this scheme, we need to be able to compare this amplification factor to the exact amplification factor of Equation 5.4. To do this, we note that (for  $\lambda \Delta t < 1$ ),

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

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.





### 5.4.5 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.

**“Minimal”.** (I can’t think of anything better to call it.) The amplification factor of a  $k$ -th order accurate Runge-Kutta scheme matches the exact amplification factor up to and including the term  $O(\Delta t^k)$  and contains no further terms.

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 first two properties of Runge-Kutta schemes are satisfied. To verify the third, recall that the amplification factor for the explicit Euler scheme is  $\sigma = 1 + \lambda \Delta t$ .

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:

$$\begin{aligned} w^{n+1} &= w^n + \lambda \Delta t w^{(1)} \\ w^{(1)} &= w^n + \frac{\lambda}{2} \Delta t w^n \end{aligned} \quad (5.8)$$

and

$$\begin{aligned} w^{n+1} &= w^n + \frac{\lambda \Delta t}{6} \left( w^n + 2w^{(1)} + 2w^{(2)} + w^{(3)} \right) \\ w^{(3)} &= w^n + \lambda \Delta t w^{(2)} \\ w^{(2)} &= w^n + \frac{\lambda}{2} \Delta t w^{(1)} \\ w^{(1)} &= w^n + \frac{\lambda}{2} \Delta t w^n \end{aligned} \quad (5.9)$$

**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.

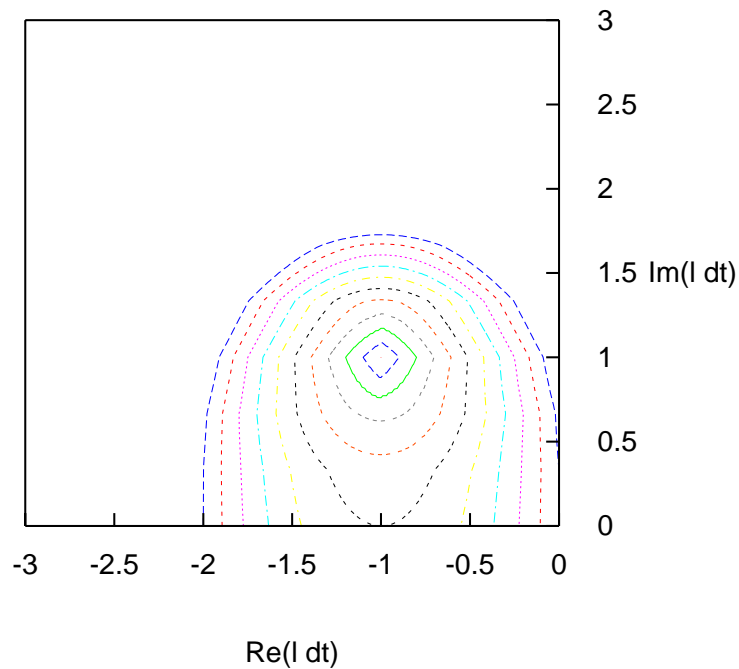
Along the imaginary axis,

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

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 stability contours for this scheme are shown below.



## 5.5 Stability Analysis for Fully-Discrete Systems

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.

## 5.6 Examples

### 5.6.1 Upwind flux with explicit Euler time advance

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

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

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

Also, we know from Section 5.4.3 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.

### 5.6.2 Centered flux with explicit Euler time advance

Using a result from Problem 5.2, we know the eigenvalues of this spatial operator:

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

These eigenvalues lie on the imaginary axis, and therefore outside the stability range of the Euler time advance scheme (see Section 5.4.3) for any time step.

## 5.7 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.<sup>3</sup> This gives us

$$\frac{\partial \bar{T}}{\partial x_i} = \alpha \frac{-\bar{T}_{i+2} + 28\bar{T}_{i+1} - 54\bar{T}_i + 28\bar{T}_{i-1} - \bar{T}_{i-2}}{24\Delta x^2}$$

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  $T_{i+\frac{1}{2}} \approx \left(\frac{\bar{T}_i + \bar{T}_{i+1}}{2}\right)$ ; 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, 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  $\frac{\mu}{\Delta x}$  before plotting).

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

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

---

<sup>3</sup>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.

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 5.9. 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 5.3 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 5.4.5 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 5.4.3) to solve the heat equation. Using results from Section 5.4.1 and 5.4.3, 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 5.7 using the fourth-order accurate spatial discretization of Problems 3.3 and 5.1 and the implicit Euler time advance scheme.

# Chapter 6

## Systems of PDE's

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, 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  $U$ .

$$\frac{d\bar{U}_{i,j,k}}{dt} + \oint_{\partial CV} \begin{pmatrix} F\hat{i} \\ G\hat{j} \\ H\hat{k} \end{pmatrix} \cdot \hat{n} dA = \bar{S}_{i,j,k} \quad (6.1)$$

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 the reacting air problem, the flux vector would be  $(\rho u, \rho u^2 + P, \rho uv, \rho uw, u(E + P), uE_v, \rho_{O_2}u, \rho_{NO}u, \rho_{OU}, \rho_{NOU})^T$ .

Systems of equations are discretized in much the same way as scalar equations. For example, we can discretize the one-dimensional equivalent of Equation 6.1 as

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} = -\frac{F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n}{\Delta x} + S_i^n \quad (6.2)$$

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 6.2; 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

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} = -\frac{F_{i+\frac{1}{2}}^{n+1} - F_{i-\frac{1}{2}}^{n+1}}{\Delta x} + S_i^{n+1} \quad (6.3)$$

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

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

and

$$\left. \frac{\partial F(U_i, U_{i+1})}{\partial t} \right|^n = \left. \frac{\partial F}{\partial U_i} \frac{\partial U_i}{\partial t} \right|^n + \left. \frac{\partial F}{\partial U_{i+1}} \frac{\partial U_{i+1}}{\partial t} \right|^n \quad (6.4)$$

by the chain rule.

We can re-write Equation 6.3 in  $\delta$ -form as

$$\begin{aligned} & \left( \frac{1}{\Delta t} + \frac{1}{\Delta x} \left. \frac{\partial F(U_i, U_{i+1})}{\partial U_i} \right|^n - \frac{1}{\Delta x} \left. \frac{\partial F(U_{i-1}, U_i)}{\partial U_i} \right|^n - \left. \frac{\partial S}{\partial U} \right|^n \right) \delta U_i^{n+1} \\ & + \frac{1}{\Delta x} \left. \frac{\partial F(U_i, U_{i+1})}{\partial U_{i+1}} \right|^n \delta U_{i+1}^{n+1} - \frac{1}{\Delta x} \left. \frac{\partial F(U_{i-1}, U_i)}{\partial U_{i-1}} \right|^n \delta U_{i-1}^{n+1} = -\frac{F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n}{\Delta x} \quad (6.5) \end{aligned}$$

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 6.4 can be written for this case as:

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

and the fully implicit discretization can be written as

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

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 6.5: 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.

## 6.1 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  $U_{i+1}$  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,

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

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.

### 6.1.1 Example: Nearly the compressible Euler equations

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

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

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:

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

**6.1.2 Example: the compressible Euler equations**

Compute the Jacobian for the following case:

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

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

## 6.2 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

$$S = \begin{pmatrix} 0 \\ 0 \\ w_O (H_{\rho_O}^0 - RT) \\ w_O \end{pmatrix}$$

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:

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

where  $H_{\rho_O}^0$  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  $\rho_{O_2}$  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 7

## Practical Aspects of Solving Poisson's Equation

### 7.1 Solving the Discrete Poisson Equation

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

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

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

$$\frac{\bar{T}_{i+1,j} - 2\bar{T}_{i,j} + \bar{T}_{i-1,j}}{\Delta x^2} + \frac{\bar{T}_{i,j+1} - 2\bar{T}_{i,j} + \bar{T}_{i,j-1}}{\Delta y^2} = S_{i,j} \quad (7.1)$$

This discretization is second-order accurate in both  $x$  and  $y$ , which you can all easily verify at this point. If we write Equation 7.2 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





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.

### 7.1.1 Iterative Methods for Poisson's Equation

We've discretized Poisson's equation as

$$\frac{\bar{T}_{i+1,j} - 2\bar{T}_{i,j} + \bar{T}_{i-1,j}}{\Delta x^2} + \frac{\bar{T}_{i,j+1} - 2\bar{T}_{i,j} + \bar{T}_{i,j-1}}{\Delta y^2} = S_{i,j} \quad (7.3)$$

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

$$\bar{T}_{i,j}^{k+1} = \frac{\Delta y^2}{2(\Delta x^2 + \Delta y^2)} \left( \bar{T}_{i+1,j}^k + \bar{T}_{i-1,j}^k \right) + \frac{\Delta x^2}{2(\Delta x^2 + \Delta y^2)} \left( \bar{T}_{i,j+1}^k + \bar{T}_{i,j-1}^k \right) - S_{i,j} \frac{\Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)} \quad (7.4)$$

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  $j$ ,<sup>1</sup> our iteration scheme would formally look like:

$$\bar{T}_{i,j}^{k+1} = \frac{\Delta y^2}{2(\Delta x^2 + \Delta y^2)} (\bar{T}_{i+1,j}^k + \bar{T}_{i-1,j}^{k+1}) + \frac{\Delta x^2}{2(\Delta x^2 + \Delta y^2)} (\bar{T}_{i,j+1}^k + \bar{T}_{i,j-1}^{k+1}) - S_{i,j} \frac{\Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)} \quad (7.5)$$

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 7.5. 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 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:

$$\begin{aligned} \delta_{i,j}^{k+1} &= \frac{\Delta y^2}{2(\Delta x^2 + \Delta y^2)} (\bar{T}_{i+1,j}^k + \bar{T}_{i-1,j}^{k+1}) + \frac{\Delta x^2}{2(\Delta x^2 + \Delta y^2)} (\bar{T}_{i,j+1}^k + \bar{T}_{i,j-1}^{k+1}) - S_{i,j} \frac{\Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)} - \bar{T}_{i,j}^k \\ \bar{T}_{i,j}^{k+1} &= \bar{T}_{i,j}^k + \omega \delta_{i,j}^{k+1} \end{aligned} \quad (7.6)$$

This scheme is referred to as successive over-relaxation (SOR); the over-relaxation parameter,  $\omega$ , 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 ( $\omega = 1.8$ ), the  $L_1$  norm of the change in solution is shown in Figure 7.1. Notice that the point Jacobi scheme appears to converge faster initially (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.

---

<sup>1</sup>That is, with loops like this:  
 for j = 1, jmax  
   for i = 1, imax  
     ...  
   end for  
end for

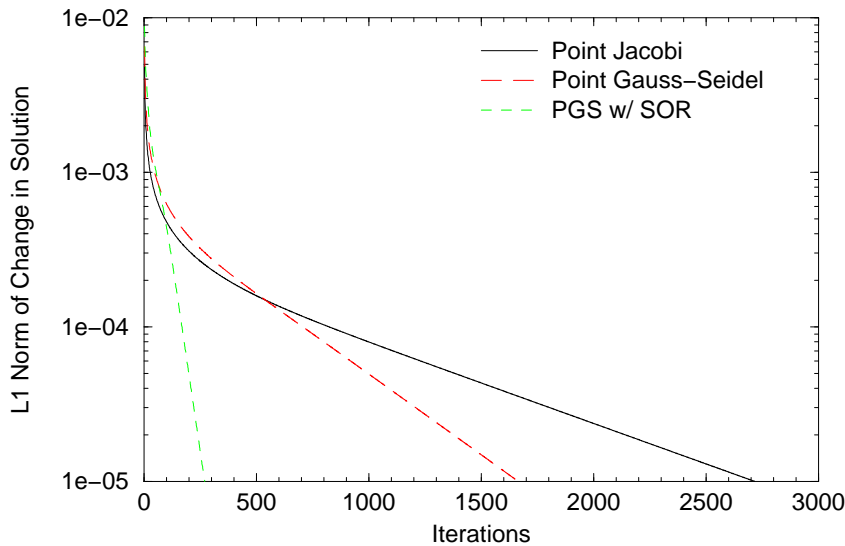


Figure 7.1: Comparison of convergence rates for point iterative schemes applied to Laplace's equation.

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

$$\begin{aligned}
 -\frac{\Delta x^2}{2(\Delta x^2 + \Delta y^2)}\delta_{i,j+1}^{k+1} + \delta_{i,j}^{k+1} \\
 -\frac{\Delta x^2}{2(\Delta x^2 + \Delta y^2)}\delta_{i,j-1}^{k+1} &= \frac{\Delta x^2}{2(\Delta x^2 + \Delta y^2)}(\bar{T}_{i,j+1}^k + \bar{T}_{i,j-1}^k) \\
 &+ \frac{\Delta y^2}{2(\Delta x^2 + \Delta y^2)}(\bar{T}_{i+1,j}^k + \bar{T}_{i-1,j}^k) - S_{i,j} \frac{\Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)} \\
 \bar{T}_{i,j}^{k+1} &= \bar{T}_{i,j}^k + \omega \delta_{i,j}^{k+1}
 \end{aligned} \tag{7.7}$$

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 7.2 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.

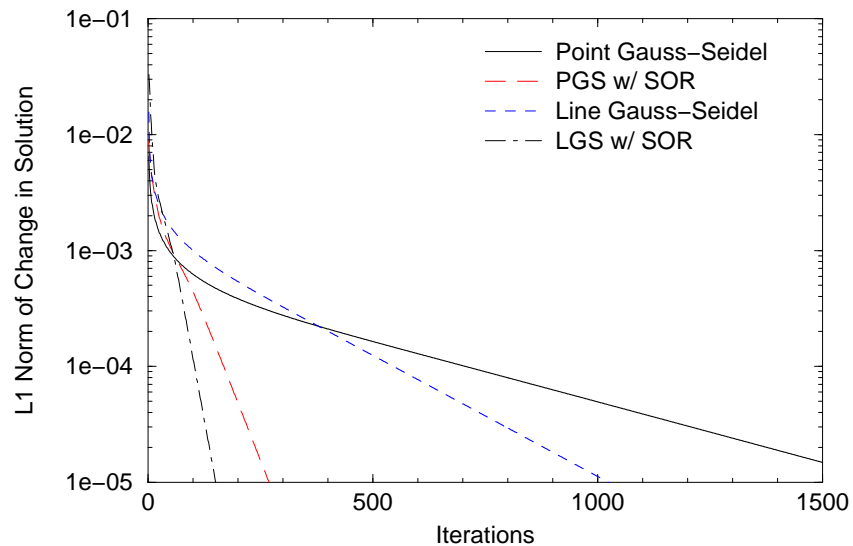


Figure 7.2: Comparison of convergence rates for point and line Gauss-Seidel iterative schemes applied to Laplace's equation.

## 7.2 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.

### 7.2.1 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 7.3, 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.

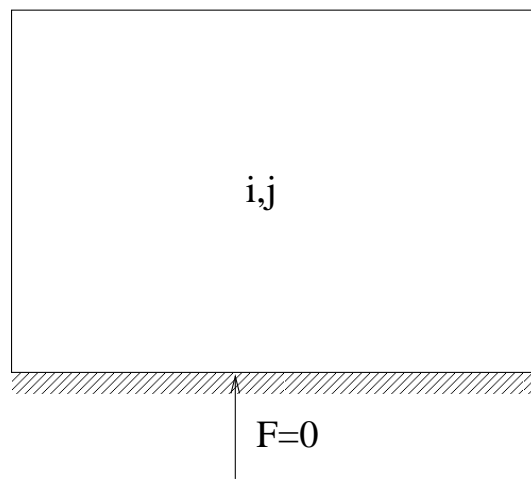


Figure 7.3: Finite volume with homogeneous Neumann boundary condition imposed along one side.

### 7.2.2 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 7.3 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 —  $T = 300\text{K}$  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.

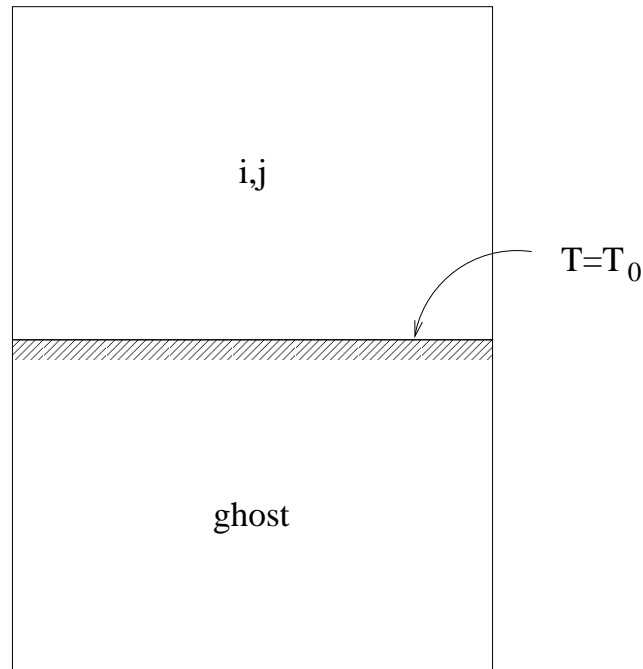


Figure 7.4: Boundary cell showing Dirichlet boundary condition and ghost cell.

### 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 8

## 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.

## 8.1 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:

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

The exact solution to this problem is

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

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.

### 8.1.1 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 (i.e.,  $T_{\frac{3}{2}} = \frac{\bar{T}_2 + \bar{T}_1}{2}$ ). 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

$$T_0 = 2g(t) - \bar{T}_1$$

### 8.1.2 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  $T_{\frac{1}{2}}^n = T(x = 0, n\Delta t) = g(n\Delta t)$ .

At what time should the boundary flux be evaluated for each stage of the two-stage Runge-Kutta scheme of Section 5.4.5? What about the four-stage scheme of Problem 5.5?

## 8.2 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 three space discretization schemes: first-order upwind, second-order upwind, and second-order centered. In each case, the CFL number is 0.75, except for the second-order upwind discretization, where  $\text{CFL} = 0.375$ . Figure 8.1 shows the results for this test. For this time advance method, both the centered and second-order upwind discretizations are unstable. Almost as bad is the first-order upwind scheme, which damps out the solution rather quickly.

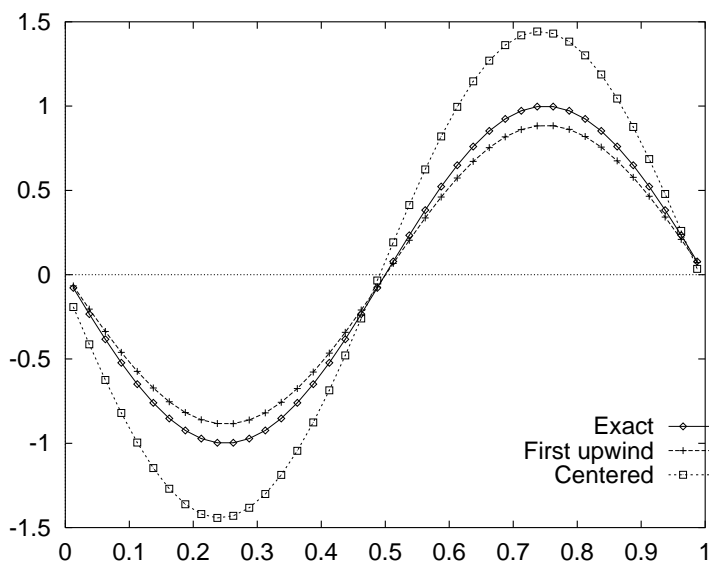


Figure 8.1: First-order time advance for the wave equation with several space discretizations

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 8.2. 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.

Now that we have a scheme<sup>1</sup> 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 8.3). 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.

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 8.4.

<sup>1</sup>Or two, if our simulations are short enough that the unstable centered scheme is okay.

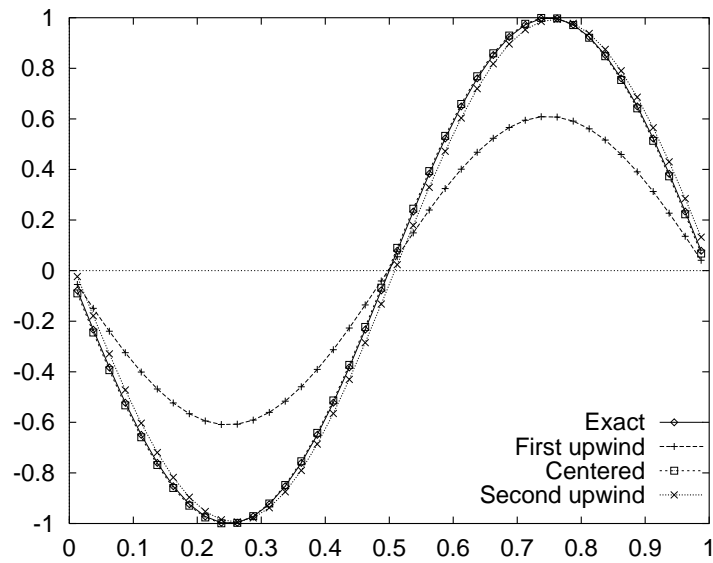


Figure 8.2: Second-order time advance for the wave equation with several space discretizations

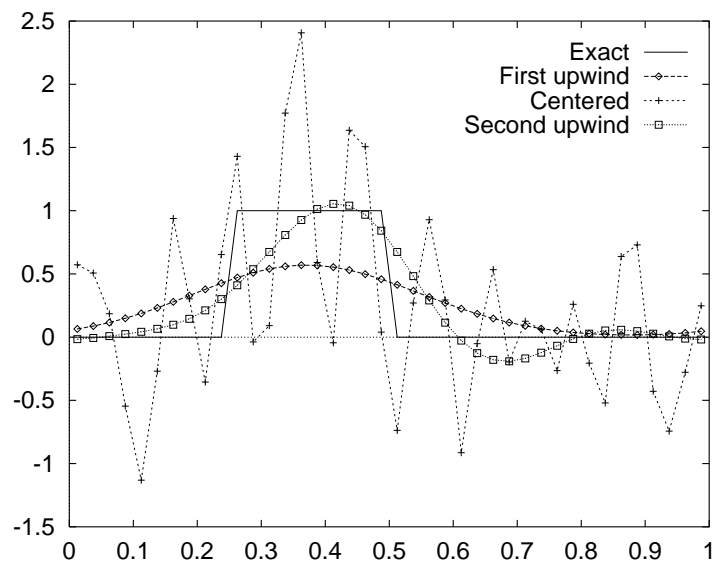


Figure 8.3: Second-order time advance for the wave equation propagating a square wave.

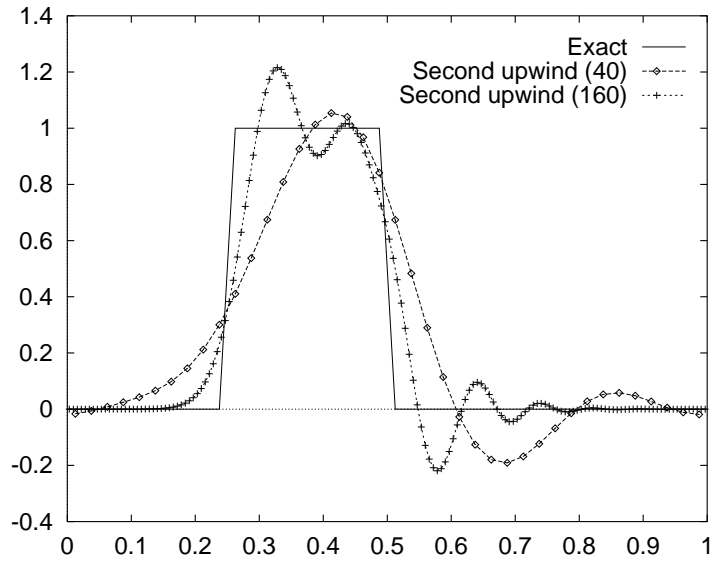


Figure 8.4: Effect of mesh refinement on square wave propagation using the second-order upwind scheme.

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

## 8.3 Advanced Schemes for the Wave Equation

### 8.3.1 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

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

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

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

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

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

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

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

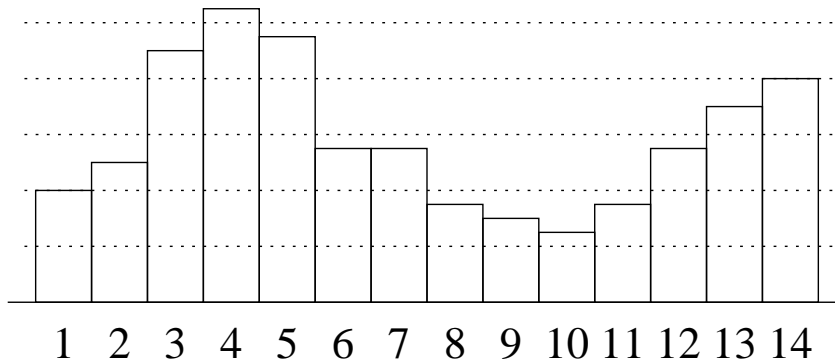


Figure 8.5: Example control-volume averaged solution

In Figure 8.5, 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 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.

### 8.3.2 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 8.5, the total variation in this solution would be

$$TV = (\bar{T}_4 - \bar{T}_1) + (\bar{T}_4 - \bar{T}_{10}) + (\bar{T}_{14} - \bar{T}_{10}) \quad (8.1)$$

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 8.1 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 with  $u = 1$ ) can be written as

$$F_{i+\frac{1}{2}} = u \left[ \bar{T}_i + \frac{\psi_{i+\frac{1}{2}}}{2} (\bar{T}_i - \bar{T}_{i-1}) \right] \quad (8.2)$$

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  $\psi$ ; 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  $\psi$  is the region outlined in Figure 8.6. Values of  $r$  less than zero indicate that CV  $i$  is a local extremum; here  $\psi$  must be zero to avoid accentuating the extremum.

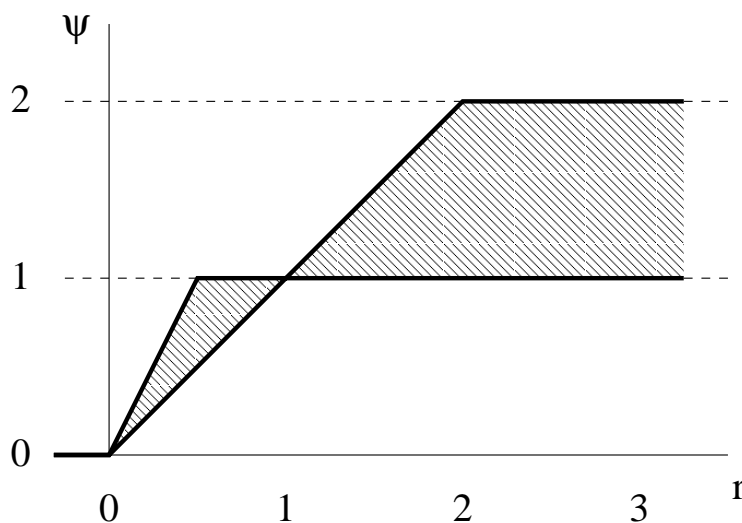


Figure 8.6: Legal values of  $\psi(r)$  for TVD schemes

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 8.5. For the maximum allowable value of  $\psi$ ,  $2r$ ,



Equation 8.2 reduces to

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

This applies only in cases where the simple extrapolation ( $\psi = 1$ ) creates a new extremum ( $r \leq \frac{1}{2}$ ). 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 8.5. 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  $r$  ( $\geq 2$ ),  $\psi$  is required to remain below 2. For  $\psi = 2$ , the TVD flux becomes

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

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  $\psi$ .

Figure 8.7 shows  $\psi$  as a function of  $r$  for three cases. The limited extrapolation scheme can be written as a TVD scheme with

$$\psi(r) = \begin{cases} 0 & \text{if } r \leq 0 \\ 2r & \text{if } r \leq \frac{1}{2} \\ 1 & \text{otherwise} \end{cases} \quad (8.3)$$

The Superbee scheme follows the top of the allowable range for  $\psi$ , making it the most “compressive”, or slope-steepening, of all possible TVD schemes.  $\psi$  for this

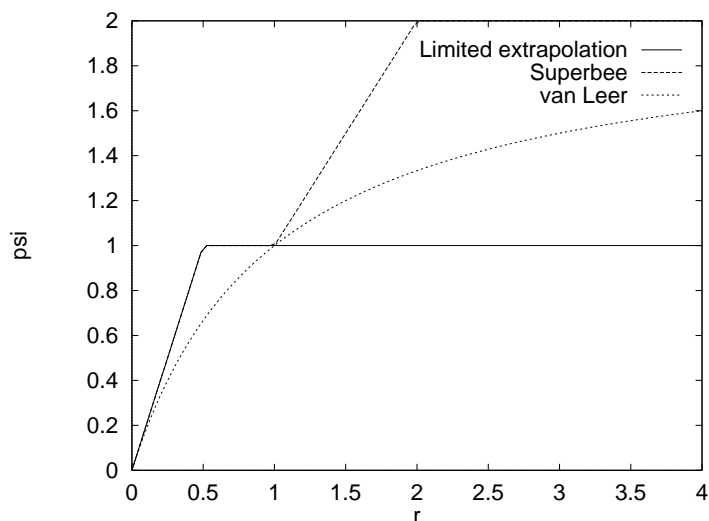


Figure 8.7: Three TVD limiters

case can be written as

$$\psi(r) = \begin{cases} 0 & \text{if } r \leq 0 \\ 2r & \text{if } 0 < r \leq \frac{1}{2} \\ 1 & \text{if } \frac{1}{2} < r \leq 1 \\ r & \text{if } 1 < r \leq 2 \\ 2 & \text{otherwise} \end{cases} \quad (8.4)$$

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

$$\psi(r) = \frac{r + |r|}{1 + r} \quad (8.5)$$

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 calculated 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  $i, j + 1$ .

### 8.3.3 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

$$F_{i+\frac{1}{2}} = u \left[ \frac{\bar{T}_i + \bar{T}_{i+1}}{2} - \left( \frac{1}{8} + \frac{u\Delta t}{2\Delta x} \right) (\bar{T}_{i+1} - \bar{T}_i) \right] \quad (8.6)$$

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.<sup>2</sup> The flux of Equation 8.6 is used to produce an interim solution at time level  $n + 1$ :

$$\tilde{T}_i^{n+1} = \tilde{T}_i^n - \frac{u\Delta t}{\Delta x} \left( F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n \right) \quad (8.7)$$

The first differences of the interim solution are computed:

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

and used to produce an anti-diffusive flux:

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

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 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

$$\bar{T}_i^{n+1} = \tilde{T}_i^{n+1} - \left( F_{i+\frac{1}{2}}^{ad} - F_{i-\frac{1}{2}}^{ad} \right) \quad (8.8)$$

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.

---

<sup>2</sup>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.

### 8.3.4 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  $O(\Delta x^{k-1})$ .

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

$$F_{i+\frac{1}{2}} = u \left( \bar{T}_i + \frac{\bar{T}_{i+1} - \bar{T}_{i-1}}{4} \right) \quad (8.9)$$

can be shown to be second-order accurate. Unfortunately, this approach allows large overshoots near discontinuities. For example, in Figure 8.5, 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 8.9 as

$$F_{i+\frac{1}{2}} = u \left[ \bar{T}_i + \frac{\Delta x}{2} \left( a \frac{\bar{T}_{i+1} - \bar{T}_i}{\Delta x} + (1-a) \frac{\bar{T}_i - \bar{T}_{i-1}}{\Delta x} \right) \right] \quad (8.10)$$

If  $a = \frac{1}{2}$ , Equations 8.9 and 8.10 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

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

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  $O(\Delta x^2)$ , 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:

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

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

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

It is easy to show that this choice of  $a$  gives

- Equation 8.9 in smooth regions, where all finite differences are bounded with mesh refinement; more precisely,  $a \rightarrow \frac{1}{2} + O(\Delta x^2)$ .

- $a \rightarrow 1 - O(\Delta x^2)$  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.
- $a \rightarrow O(\Delta x^2)$  when a discontinuity exists between control volumes  $i$  and  $i + 1$ ; this is a one-sided extrapolation from the other side.

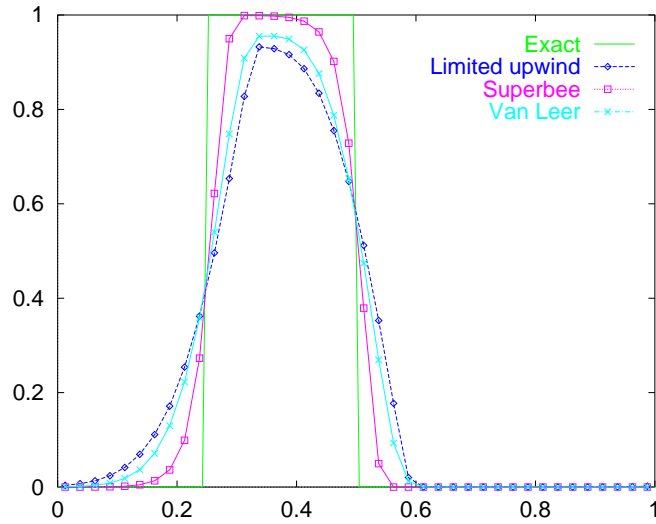
Note that a particularly clever choice of  $a$  would give you a third-order accurate flux at the interface for smooth solutions. I personally don't use this choice. My interest in this problem comes from the area of reconstruction on multidimensional unstructured meshes, where the extension of this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

### 8.3.5 Sample Calculations

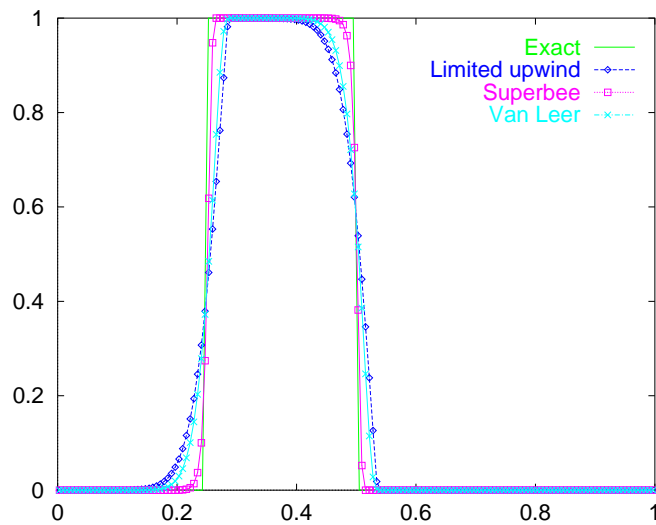
Figures 8.8–8.10 show the results of applying all these schemes to the wave equation with both square wave and sine wave initial data. In all cases, the boundary conditions are periodic and the solution is advanced in time until the wave has traveled around the mesh exactly once.

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 ENO schemes, because I understand exactly how to generalize them to arbitrary meshes. While generalization is also possible with TVD and FCT schemes, these generalizations all seem more forced to me.

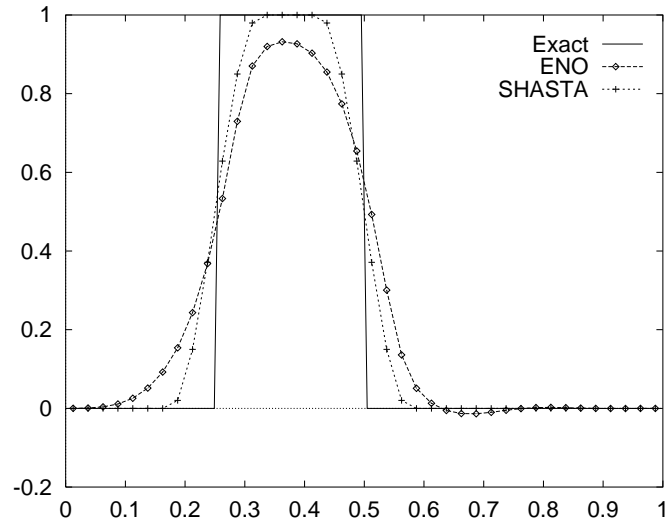


(a) Coarse mesh

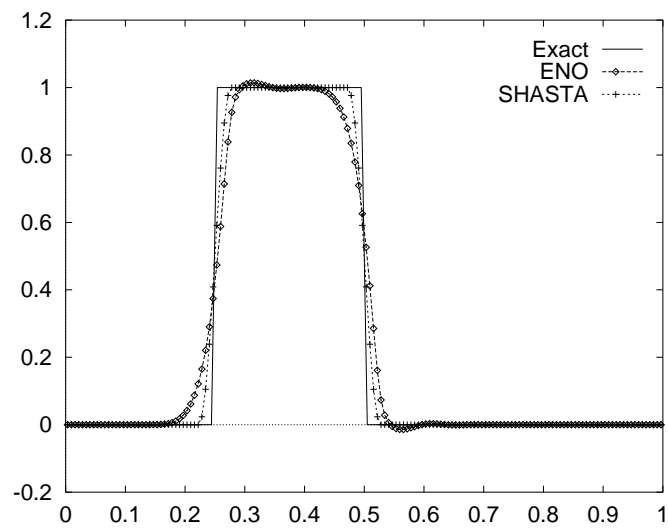


(b) Fine mesh

Figure 8.8: Upwind TVD schemes (square wave)



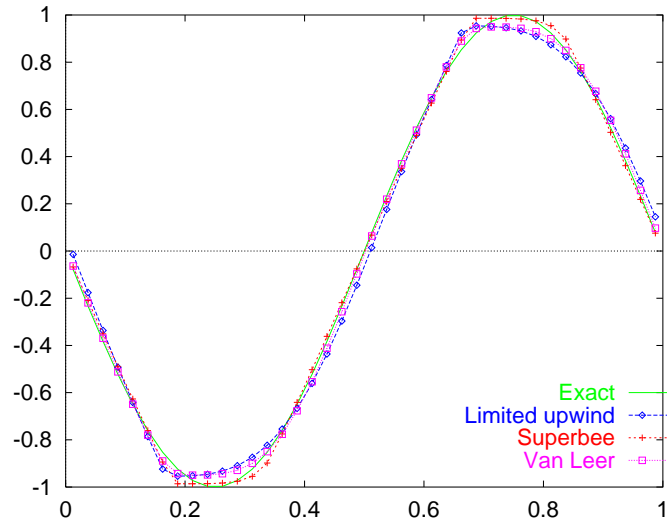
(a) Coarse mesh



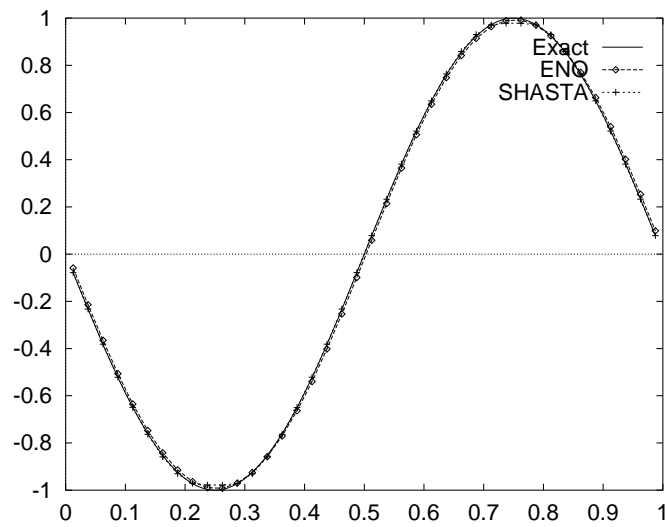
(b) Fine mesh

Figure 8.9: ENO and FCT schemes (square wave)





(a) TVD schemes



(b) ENO and FCT schemes

Figure 8.10: Propagation of a smooth solution (sine wave)



# Chapter 9

## 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:

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

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:

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

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.

$$\frac{d\bar{T}_{i,j}}{dt} + \frac{1}{\Delta x} \left( uT - \frac{1}{Re \cdot Pr} \frac{\partial T}{\partial x} \right)_{i-\frac{1}{2},j}^{i+\frac{1}{2},j} + \frac{1}{\Delta y} \left( vT - \frac{1}{Re \cdot Pr} \frac{\partial T}{\partial y} \right)_{i,j-\frac{1}{2}}^{i,j+\frac{1}{2}} = S_{i,j} \quad (9.1)$$

## **9.1 Simple Discretization of the Incompressible Energy Equation**

## 9.2 Time Discretization of the Energy Equation

### 9.2.1 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:

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

In practice, we will typically want to write this in what is called  $\delta$ -form by replacing  $T_{i,j}^{n+1} \equiv T_{i,j}^n + \delta T_{i,j}^{n+1}$  and simplifying.  $\delta$ -form is much more convenient near steady state, where round off errors in the calculation of  $T$  can easily exceed  $\delta T$ , the change in  $T$  from one time level to the next.

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

$$\begin{aligned}
&= -\frac{1}{\Delta x} \left( \frac{u_{i+1,j} \bar{T}_{i+1,j}^n - u_{i-1,j} \bar{T}_{i-1,j}^n}{2} \right. \\
&\quad \left. - \frac{1}{Re \cdot Pr} \left( \frac{\bar{T}_{i+1,j}^n - 2\bar{T}_{i,j}^n + \bar{T}_{i-1,j}^n}{\Delta x} \right) \right) \\
&- \frac{1}{\Delta y} \left( \frac{v_{i,j+1} \bar{T}_{i,j+1}^n - v_{i,j-1} \bar{T}_{i,j-1}^n}{2} \right. \\
&\quad \left. - \frac{1}{Re \cdot Pr} \left( \frac{\bar{T}_{i,j+1}^n - 2\bar{T}_{i,j}^n + \bar{T}_{i,j-1}^n}{\Delta y} \right) \right) \\
&+ S_{i,j}
\end{aligned} \tag{9.2}$$

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

### 9.2.2 Trapezoidal time advance applied to the energy equation

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,

$$\begin{aligned} F_{i+\frac{1}{2},j}^{n+1} &= F(U_{i,j}^{n+1}, U_{i+1,j}^{n+1}) \\ &= F(U_{i,j}^n + \delta U_{i,j}, U_{i+1,j}^n + \delta U_{i+1,j}) \end{aligned} \tag{9.3}$$



$$= F(U_{i,j}^n, U_{i+1,j}^n) + \frac{\partial F_{i+\frac{1}{2},j}^n}{\partial U_{i,j}} \delta U_{i,j} + \frac{\partial F_{i+\frac{1}{2},j}^n}{\partial U_{i+1,j}} \delta U_{i+1,j} + O((\delta U)^2) \quad (9.4)$$

where a two-variable Taylor series expansion is used between Equations 9.3 and 9.4.

This result requires that we calculate flux Jacobians: derivatives of the fluxes with respect to the unknowns in nearby control volumes; see Section 6.1 for more information about how to do this. The Jacobians we obtain are:

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

where the inviscid and viscous parts have been combined.

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

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

Multiplying by  $\Delta t$  and labeling terms in an obvious way, we can write:

$$\begin{aligned} (I + \Delta t B_x + \Delta t B_y) \delta U_{i,j} \\ + \Delta t C_x \delta U_{i+1,j} + \Delta t A_x \delta U_{i-1,j} = -\Delta t \frac{F_{i+\frac{1}{2},j}^n - F_{i-\frac{1}{2},j}^n}{\Delta x} \quad (9.5) \\ + \Delta t C_y \delta U_{i,j+1} + \Delta t A_y \delta U_{i,j-1} - \Delta t \frac{G_{i,j+\frac{1}{2}}^n - G_{i,j-\frac{1}{2}}^n}{\Delta y} \end{aligned}$$

### 9.3 Approximate Factorization

We can re-write Equation 9.5 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:

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



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.

### 9.4.1 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:

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

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

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

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

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

Therefore,

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

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.

### 9.4.2 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

$$\begin{aligned} u_{i,0} &= -u_{i,1} \\ v_{i,0} &= -v_{i,1} \\ P_{i,0} &= +P_{i,1} \end{aligned} \tag{9.7}$$

### 9.4.3 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

$$\begin{aligned} u_{i,0} &= 2u_{\text{wall}} - u_{i,1} \\ v_{i,0} &= -v_{i,1} \\ P_{i,0} &= +P_{i,1} \end{aligned} \tag{9.8}$$

### 9.4.4 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

$$\begin{aligned} u_{i,0} &= -u_{i,1} \\ v_{i,0} &= 2v(x,0) - v_{i,1} \\ P_{i,0} &= +P_{i,1} \end{aligned} \tag{9.9}$$

### 9.4.5 Implicit boundary condition implementation

If we construct the tri-diagonal system for the  $i^{\text{th}}$  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).

### 9.4.6 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:

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

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

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

$$\begin{aligned} v_{0,j} &= -v_{1,j} \\ P_{0,j} &= P_{1,j} - \Delta x \frac{\partial P}{\partial x_{in}} \end{aligned} \quad (9.10)$$

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.

### 9.4.7 Outflow boundaries

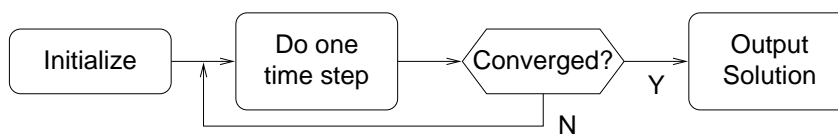
For outflow boundaries, we can specify the pressure  $P_{out}$  and assume fully-developed flow (no stream-wise velocity gradient). These conditions are actually quite reasonable. We often want to control the back pressure for internal flows, and a flow that is not fully-developed at the exit may in 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

$$\begin{aligned} u_{i_{\max}+1,j} &= u_{i_{\max},j} \\ v_{i_{\max}+1,j} &= v_{i_{\max},j} \\ P_{i_{\max}+1,j} &= 2P_{out} - P_{i_{\max},j} \end{aligned} \quad (9.11)$$

The implicit version of Equation 9.11 is

## 9.5 Outline of Navier-Stokes Code



### Initialize

Set boundary conditions

```
Set parameters (Re,  $\Delta t$ , 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

elliptic		$< 0$
parabolic	if $B^2 - 4AC$	$= 0$
hyperbolic		$> 0$

### 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:

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

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

A similar expansion can be written in two dimensions:

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

### 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  $\lambda_k$  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  $\lambda_k$  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  $B_p(\dots, a_{-2}, a_{-1}, a_0, a_1, a_2, \dots)$  of size  $i_{\max}$  (rows numbered from  $i = 1$ ) has a complete eigensystem. The right eigenvectors are of the form

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

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

$$\lambda_k = \sum a_j e^{jI\phi_k} \quad (\text{B.4})$$

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  $B_p X_k = \lambda_k X_k$ . Substituting the expressions above, we have, for a general vector element  $i$ :

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

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

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

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.





```

! 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 possible 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 in
   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, and
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<sup>1</sup>, 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  $26 \times 38$ , 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.

---

<sup>1</sup>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;<sup>2</sup> 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 ??). 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:

---

<sup>2</sup>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.<sup>3</sup>

### D.3.4 Higher-dimensional Arrays

Let’s suppose that you want to store three  $3 \times 3$  matrices for every cell in each column of your mesh.<sup>4</sup> 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  $3 \times 3$  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  $3 \times 3$  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.

---

<sup>3</sup>Fortran 90 introduced a new interface for array slicing into Fortran that does this copying for you.

<sup>4</sup>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

## Validating CFD Programs

Once you have written a CFD program that runs to completion, there is essentially zero chance that it is correct without further testing and debugging. Inevitably, there will be errors. The validation phase of a CFD project consists of extensive testing to verify that the computer program actually produces correct solutions to the mathematical model of the physical problem.

Generally, a good validation plan will begin by verifying that the program correctly computes flux integrals, then move on to verify boundary conditions, time advance, and convergence properties of the code.

During the course of the term, several modeling and validation assignments will be given. This handout is intended to give you general information about what I expect you to do for each of these assignments.

For each assignment, the description of a physical problem in fluid mechanics or heat transfer will be distributed. You will have a week to consider what physical phenomena are important in solving the problem and to decide what test problems you would use to validate a program written to solve the problem. Bring with you to class *two copies* of your write-up for the problem, one to submit at the beginning of class and one to refer to during group discussion of the problem in class. After you have discussed physical modeling and validation for the problem in small groups, each group will submit their combined results and we will discuss the results in class. Marks will be assigned both for individual and group results.<sup>1</sup>

---

<sup>1</sup>Note that I will pick the groups for these discussions, and the groups will be different for each assignment.

The following class period, I will hand out a summary of the class's combined wisdom on modeling and validation for the problem.

Your write-up should consist of two sections: modeling, and validation protocol.

## E.1 Modeling

In this part of your write-up, you should describe which physical phenomena are relevant for solution of the problem and which can be safely neglected. This is of course a judgment call in many cases, so sentence or two justifying your choices may go a long way to convincing others that you are correct. For example, for the tire incinerator problem discussed during the first week of class, you might have something like:

- Viscosity is important to correctly simulate convection heat transfer and flow separation.
- The flow will likely be turbulent despite low flow velocity because of the large size of the physical domain.
- The flow will be incompressible in the sense that Mach number effects will be unimportant, but density variations will still be significant because of large differences in temperature.
- Heat release due to combustion must be included, as it will dominate the energy balance of the flow.
- Resolution of the chemical reaction zone will be assumed *not* to be necessary; instead heat release will be modeled using a volumetric source term.

There are doubtless many others that could be added, but hopefully you get the idea. I am specifically *not* interested in details at the level of “I would use the  $k - \epsilon$  turbulence model” or “Chemical kinetics would be modeled using rates determined by Smith, Jones, and Park”. Questions such as these are always difficult to answer; I don't expect you to spend time worrying about such questions for these assignments.

## E.2 Validation

Okay, so now you've decided what physics to simulate in the final program. Now you need to make a list of particular test cases to run to test each of these things. Several points to keep in mind:

- These need not be physical flow problems; they can instead be designed to test as little as a single subroutine from the program.
- No test case is too simple if it checks any untested piece of physics or any untested interaction between separate pieces of physics.
- Geometry can differ from case to case just as the flow conditions can.

For each case, a one or two sentence description of what the case tests should be included, to make the purpose of non-obvious tests more clear. Some general sorts of tests that are nearly always worth doing:

- Validate your flux integrals using some prescribed initial solution with a known flux integral.
- Symmetry checks are often useful, as are problems in which one dimension is redundant (for example, using a 3D code to solve a 2D channel flow problem with multiple cells (and periodic boundary conditions) in the third dimension).
- Sanity checks, such as checking for correct trends in the solution with changes in boundary or initial conditions. For example, for the tire incinerator, higher heat release from combustion should tend to increase both temperature in the incinerator and heat flux through the walls.



# Appendix F

## 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.