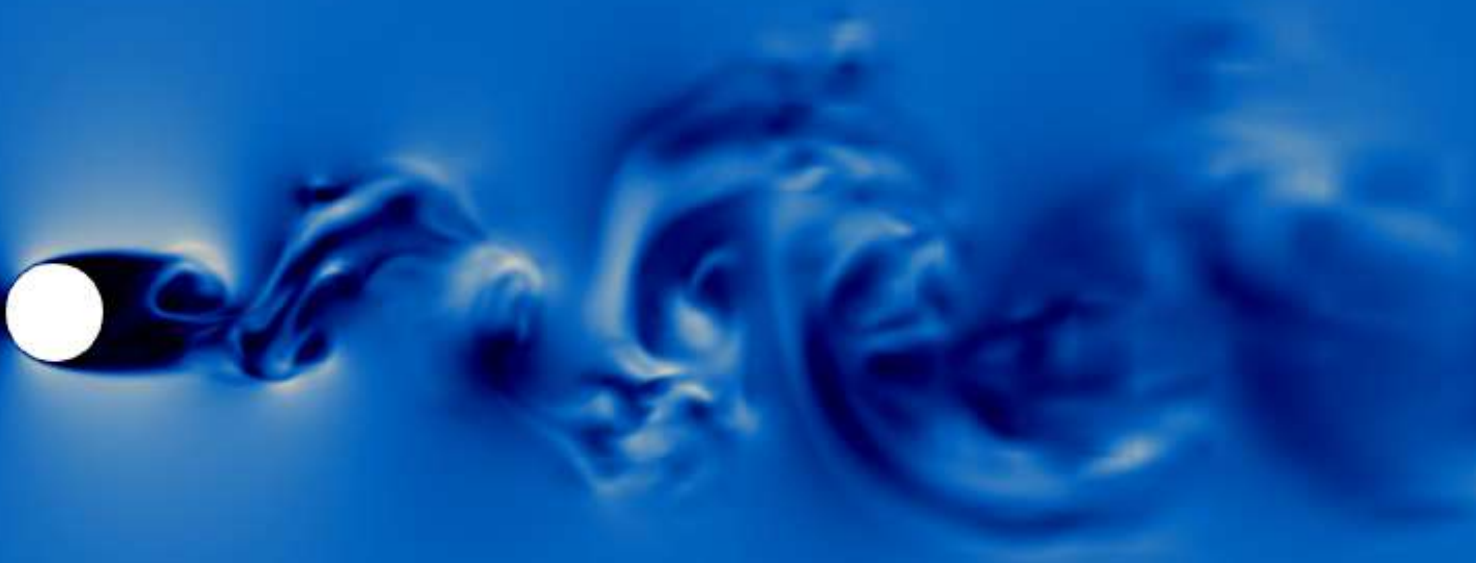


Computational Fluid Dynamics

An Open Source Approach



Brian C. Vermeire

Carlos A. Pereira

Hamidreza Karbasian

Computational Fluid Dynamics

An Open Source Approach

Copyright © 2020 Brian C. Vermeire

PUBLISHED BY CONCORDIA UNIVERSITY

BOOK-WEBSITE.COM

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <http://creativecommons.org/licenses/by-nc/3.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, October 2020

Foreword

This book evolved out of my lecture notes for the undergraduate and graduate computational fluid dynamics courses that I teach at Concordia University. In combination with an open-educational resources grant from the library, it is provided to you completely free of charge. It uses entirely open-source tools including Python, Jupyter Notebooks, SU2, Gmsh, Paraview, L^AT_EX, and many others. This means that all of the examples and applications can be run on your computer using any common operating system and without purchasing any licenses - CFD for free!

The book itself is even open-source, available as a public repository on Gitlab. As such, you may want to think of this book more as a software development project, rather than a conventional hard cover textbook. All of the content needed to run the chapter examples and final applications are stored in the repository, and you can even contribute to the book and its examples via a pull request if you think of a useful addition. Just like any software development project, this book may contain a few “bugs”, mostly in the form of minor typos. If you find one of these please feel free to give back and submit a pull request to correct them.

In terms of content, the book is designed with enough material to cover an advanced undergraduate course, or an introductory course for graduate students. For an undergraduate course I recommend a more hands-on computer lab experience, focusing on Part 1, some of Part 2, and Part 3. In Part 2 I find it useful to cover finite difference methods, consistency, stability, convergence, time stepping, iterative methods, and then return to introduce finite volume methods. Part 3 is designed as a bi-weekly computer lab, where students get hands-on experience with practical CFD simulations. For a graduate course I recommend focusing on Parts 1 and 2, with a final project to write a two-dimensional compressible solver for lid driven cavity flow. If you are studying CFD on your own then I recommend covering the whole book.

Finally, I would like to thank my undergraduate and graduate students over the years for their useful discussions and contributions to the core ideas in this book. In particular, I need to acknowledge my co-authors Carlos and Hamid for their hard work in developing the first version. I also thank the students in my classes for using the first “experimental” editions and their useful feedback. Finally, I would like to thank you the reader for your interest in this project and for learning CFD.

Dr. Brian C. Vermeire

Contents

I	Part 1: Physics	
1	Conservation Laws	13
1.1	Reynolds Transport Theorem	13
2	The Navier Stokes Equations	15
2.1	Integral Form	15
2.1.1	Conservation of Mass	15
2.1.2	Conservation of Momentum	15
2.1.3	Conservation of Energy	16
2.1.4	Compact Integral Form	17
2.2	Divergence Form	18
2.2.1	Conservation of Mass	18
2.2.2	Conservation of Momentum	18
2.2.3	Conservation of Energy	19
2.2.4	Compact Divergence Form	19
3	Simplified Systems	20
3.1	Euler Equations	21
3.2	Linear Advection	21
3.3	Burgers Equation	22
3.4	Linear Diffusion	23
3.5	PDE Classification	25
3.5.1	First Order Equations	25
3.5.2	Second Order Equations	25

4	Turbulence	26
4.1	Turbulence Theory	26
4.1.1	Introduction to Chaos	27
4.1.2	Chaos and Navier-Stokes	28
4.1.3	The Energy Cascade	30
4.2	Reynolds Averaging	32
4.2.1	The Reynolds Averaged Navier-Stokes Equations	34
4.2.2	The Reynolds Stresses	38
4.2.3	The RANS Closure Problem	39
4.3	Turbulence Modelling	42
4.3.1	The Boussinesq Hypothesis	42
4.3.2	The Mixing Length Model	43
4.3.3	The Spalart-Allmaras Model	44
4.3.4	The $k-\epsilon$ Model	45
4.3.5	The $k-\omega$ Model	46
4.3.6	Summary of Turbulence Models	47
5	Boundary Conditions	49
5.1	Wall Boundaries	49
5.1.1	Wall-Bounded Turbulence	49

II

Part 2: Numerics

6	Taylor-Series	53
7	Finite Difference Methods	56
7.1	The First Derivative	56
7.2	A General Approach	58
7.2.1	Step 1: Generate the Taylor Series	58
7.2.2	Step 2: Rearrange the Taylor Series	58
7.2.3	Step 3: Determine a Suitable Combination	59
7.2.4	Step 3: Combine the Schemes	59
7.3	The Second Derivative	59
7.4	Example Applications	60
7.4.1	Linear Advection	60
7.4.2	Burgers Equation	62
7.4.3	Linear Diffusion	62
8	Finite Volume Methods	65
8.1	Derivation	66
8.2	The Riemann Problem	67
8.3	Example Applications	68
8.3.1	Linear Advection	68
8.3.2	Burgers Equation	70
8.3.3	Linear Diffusion	71

8.4	Linear Hyperbolic Problems	72
8.4.1	Linear Hyperbolic Systems	73
8.4.2	The Riemann Problem	75
8.5	Nonlinear Hyperbolic Problems	80
8.5.1	Nonlinear Hyperbolic Systems	83
8.5.2	The Riemann Problem for the Euler equations	84
8.5.3	A Riemann Solver for the Euler Equations	86
8.6	MUSCL Schemes	88
8.6.1	Second-Order Upwind Scheme for Linear Advection	89
8.6.2	Total Variation Diminishing	90
8.6.3	Limiters	90
8.6.4	Numerical Examples	91
9	Consistency, Stability, Convergence	95
9.1	Consistency	95
9.2	Stability	96
9.2.1	Explicit Linear Advection	97
9.2.2	Implicit Linear Advection	99
9.2.3	Explicit Linear Diffusion	100
9.2.4	Implicit Linear Diffusion	102
9.3	Convergence	103
10	Spectral Properties	105
10.1	Dissipation Error	105
10.2	Dispersion Error	107
11	Modified Equation Analysis	110
11.1	Linear Advection	111
11.2	General Observations	112
12	Time-Stepping	113
12.1	Explicit	113
12.1.1	Forward Euler	114
12.1.2	Heun's Method	114
12.1.3	Midpoint Method	115
12.1.4	Runge-Kutta Methods	115
12.2	Implicit	117
12.2.1	Implicit Linear Advection	117
12.2.2	Implicit Linear Diffusion	119
12.2.3	Implicit Burgers Equation	119
13	Iterative Methods	121
13.1	Gaussian Elimination	121
13.2	Jacobi Iteration	122
13.3	Gauss Seidel Iteration	124
13.4	Successive Over-Relaxation	126

13.5	Assessing Convergence	126
13.6	Multigrid	127
14	Applications	130
14.1	An Euler Solver	130
14.2	A Navier-Stokes Solver	132

III

Part 3: Applications

15	Introduction	137
16	Inviscid NACA 0012	139
16.0.1	Load the Solution File	140
16.0.2	Visualize the Mesh	140
16.0.3	Visualize Pressure Contours	141
16.0.4	Visualize the Pressure Coefficient	143
16.0.5	Aerodynamic Forces	147
17	Supersonic Wedge	149
17.0.1	Load the Solution File:	150
17.0.2	Visualize the Mesh	151
17.0.3	Visualize Pressure and Mach Contours	151
17.0.4	Plotting a variable over an arbitrary line	157
18	Inviscid ONERA M6	161
18.0.1	Load the Solution File:	162
18.0.2	Visualize the Mesh	163
18.0.3	Pressure Coefficient and Mach Number Contours	163
18.0.4	Comparison of Convergence Rates	167
19	Laminar Cylinder	170
19.0.1	Load the Solution File:	172
19.0.2	Visualize the Mesh	172
19.0.3	Visualize Pressure Contour and Much Number Contour	172
19.0.4	Streamlines and Separation Length	175
19.0.5	Shedding Frequency	178
20	Turbulent ONERA M6	183
20.0.1	Load the Solution File:	184
20.0.2	Visualize the Mesh	185
20.0.3	Visualize Pressure Coefficient at Different Stations	185
20.0.4	Visualize Pressure Coefficient (Alternative Method) Exporting .csv file at Different Stations	191
21	Mesh Generation Using Gmsh	194
22	Shock Waves	205
22.0.1	Load the Solution File:	207

22.0.2	Visualize the Mesh	207
22.0.3	Visualize Pressure, Temperature and Mach Contours	208
22.0.4	Plotting non-dimensional variables along the centerline	210

Bibliography 214

Articles 214

Books 215



Part 1: Physics

1	Conservation Laws	13
1.1	Reynolds Transport Theorem	
2	The Navier Stokes Equations	15
2.1	Integral Form	
2.2	Divergence Form	
3	Simplified Systems	20
3.1	Euler Equations	
3.2	Linear Advection	
3.3	Burgers Equation	
3.4	Linear Diffusion	
3.5	PDE Classification	
4	Turbulence	26
4.1	Turbulence Theory	
4.2	Reynolds Averaging	
4.3	Turbulence Modelling	
5	Boundary Conditions	49
5.1	Wall Boundaries	

1. Conservation Laws

Some of the most powerful tools in classical mechanics, including fluid mechanics, are conservation laws. Arising from the profound physical insights of Newton, Leibniz, and others, these laws ensure that the total amount of certain physical quantities within a volume are conserved. For example, conservation of mass ensures that the total mass of a system remains constant, conservation of momentum ensures that the total momentum of a system remains constant, and conservation of energy ensures that the total energy of a system remains constant. While these concepts can be applied by a high-school student for simple systems, such as elastic/inelastic collisions between particles, their application to fluid mechanics is less trivial. Nevertheless, the fundamental concepts of conservation of mass, momentum, and energy still apply to fluids just as well as they do to individual particles, it is only the mathematics that becomes more complex. It is expected that students reading this book have already taken an undergraduate course in fluid mechanics and are familiar with conservation laws. Nevertheless, this chapter reviews these concepts for completeness, and to establish the notation used in the rest of the book. Some other useful references include [7, 14, 24].

1.1 Reynolds Transport Theorem

Before tackling conservation of mass, momentum, and energy in their entirety, we will first consider an arbitrary conserved *extensive* quantity $U_{System} = U_{System}(t)$ of a moving system of fluid, which has a related quantity per unit volume $u = u(\vec{x}, t)$, where \vec{x} and t are the spatial coordinate and time, respectively. For example, if the extensive property is mass, then the volumetric property is mass per unit volume, or density. We start by imagining a stationary control volume (CV) such as the one in Figure 1.1, denoted by Ω , that is the same shape as the fluid system at some time t . We also denote the surface of this volume by S and the outward pointing normal vector on this surface by \hat{n} . After some amount of time dt , we can imagine that the initial system of fluid embedded within the control volume will move to a deformed position and shape at $t + dt$, while the control volume remains fixed by definition. In this manner, the fluid contained by the system will be equal to that of the control volume at time t , plus any incoming or outgoing fluid due to the motion of the system boundaries as it travels with the flow.

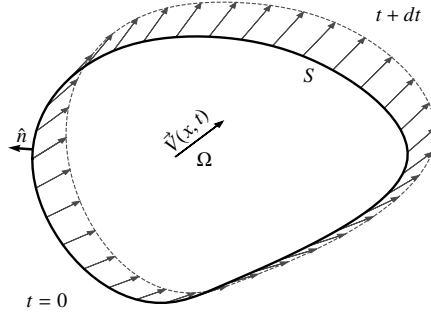


Figure 1.1: Arbitrary flow pattern on an arbitrary volume

Looking at Figure 1.1, there are two ways that U_{System} will change with time. Either via a change of U_{CV} within the control volume it overlaps with, or by some amount of the conserved quantity crossing the boundary of the system as it moves. We start by getting the total amount within the control volume $U_{CV}(t)$ by simply adding up, or integrating, u over it. This can be written as

$$U_{CV}(t) = \int_{\Omega} u d\vec{x}, \quad (1.1)$$

noticing that the dependence on space is lost after integration. The time derivative of this is associated with the rate of change of the conserved variable within the control volume itself

$$\frac{dU_{CV}}{dt} = \frac{d}{dt} \int_{\Omega} u d\vec{x}. \quad (1.2)$$

The second term to be addressed is the change in U_{System} due to the fluid crossing its boundaries.

As noted previously, the second way for U_{System} to change with time is by fluid crossing the surface S as the fluid system advances to $t + dt$. In order for fluid to cross the surface, it must be moving *normal* to it, otherwise, it will just move along the surface and not enter the control volume. Hence, we can get the velocity component normal to the surface at any point via the normal vector. Then we can get the total amount of u crossing the surface S by adding up, or integrating, the normal flux at every point on the surface

$$F(t) = \oint_S u(\vec{v} \cdot \hat{n}) ds, \quad (1.3)$$

where $F = F(t)$ is the total rate the conserved quantity U enters/leaves the control volume across the surface.

To create our conservation law, we combine the concepts in Equations 1.2 and 1.3. We note that the rate of change of the total amount of the conserved variable within the system U_{System} is equal to the rate at which the conserved variable changes within the control volume and the rate it enters/leaves across the surface of the system. Mathematically, this can be written as

$$\frac{dU_{System}}{dt} = \frac{d}{dt} \int_{\Omega} u d\vec{x} + \oint_S u(\vec{v} \cdot \hat{n}) ds, \quad (1.4)$$

noting that the positive in front of the surface term is due to our normal vector being outward pointing. Known as Reynolds Transport Theorem, this will be the foundation for deriving our conservation of mass, momentum, and energy equations for fluid flows.

2. The Navier Stokes Equations

2.1 Integral Form

2.1.1 Conservation of Mass

When considering the mass $m = m(t)$ of our system, the mass per unit volume is the density, denoted by $\rho = \rho(\vec{x}, t)$. Since the total mass must be conserved, the internal change within the control volume must equal the rate it leaves/enters over S . Hence,

$$\frac{dm}{dt} = \frac{d}{dt} \int_{\Omega} \rho d\vec{x} + \oint_S \rho \vec{v} \cdot \hat{n} ds = 0, \quad (2.1)$$

and conservation of mass can be written as

$$\frac{d}{dt} \int_{\Omega} \rho d\vec{x} + \oint_S \rho \vec{v} \cdot \hat{n} ds = 0. \quad (2.2)$$

2.1.2 Conservation of Momentum

From Newton's second-law, we know that the time rate of change of the total momentum of the system is equal to the sum of all forces acting on it. Hence

$$\sum \vec{F} = \frac{d(m\vec{v})}{dt}, \quad (2.3)$$

where the product $m\vec{v}$ is the total momentum of the system. Noting that the momentum per unit volume is $\rho\vec{v}$, and using Reynolds transport theorem, we obtain

$$\frac{d}{dt} \int_{\Omega} \rho \vec{v} d\vec{x} + \oint_S \rho \vec{v} (\vec{v} \cdot \hat{n}) ds = \sum \vec{F}, \quad (2.4)$$

which requires knowledge of the forces that will act on the system at any given time, which can be split into surface and body terms. In the current context, only surface terms are considered and body terms, such as gravitational forces, are neglected.

The first of the surface forces is due to the pressure surrounding the system. Since pressure acts normal to a surface, the total pressure force \vec{F}_P can be obtained via integration along the surface. Hence,

$$\vec{F}_P = \oint_S -p\hat{n}ds, \quad (2.5)$$

where p is the pressure and the negative is included since pressure exerts a force inwards, but our normal vector is defined as outwards. The second set of surface terms is due to the effects of viscosity. To account for these we introduce the Cauchy stress tensor τ , which for Newtonian fluids is [7]

$$\tau = \mu \left(\nabla \vec{v} + (\nabla \vec{v})^T \right) - \frac{2}{3} \mu (\nabla \cdot \vec{v}) \mathbf{I}, \quad (2.6)$$

where μ is the dynamic viscosity and \mathbf{I} is an identity matrix. Whereas pressure acts normal to the control volume surface, viscous effects act parallel to it. Hence, the total viscous force \vec{F}_v can be obtained via integration

$$\vec{F}_v = \oint_S \tau \cdot \hat{n}ds. \quad (2.7)$$

With the inviscid and viscous forces determined, conservation of momentum can be written as

$$\frac{d}{dt} \int_{\Omega} \rho \vec{v} d\vec{x} + \oint_S \rho \vec{v} (\vec{v} \cdot \hat{n}) ds = \oint_S -P \hat{n} ds + \oint_S \tau \cdot \hat{n} ds, \quad (2.8)$$

which is commonly written by grouping all of the surface integral terms

$$\frac{d}{dt} \int_{\Omega} \rho \vec{v} d\vec{x} + \oint_S [\rho \vec{v} \otimes \vec{v} - \sigma] \cdot \hat{n} ds = 0, \quad (2.9)$$

where

$$\sigma = -p\mathbf{I} + \tau. \quad (2.10)$$

2.1.3 Conservation of Energy

From conservation of energy, the rate of change of energy within the system is equal to the rate of heat added to the system less the rate work done by the system on its surroundings. Hence,

$$\frac{dE}{dt} = \dot{Q} - \dot{W}, \quad (2.11)$$

where E is the energy in the system, \dot{Q} is heat rate, and \dot{W} is work rate. In this case, the energy per unit volume is ρe where

$$e = c_v T + \frac{1}{2} \vec{v} \cdot \vec{v}, \quad (2.12)$$

is the specific energy, c_v is the specific heat at constant volume, and T is the temperature. Using Reynolds transport theorem, we have

$$\frac{d}{dt} \int_{\Omega} \rho e d\vec{x} + \oint_S \rho e (\vec{v} \cdot \hat{n}) ds = \dot{Q} - \dot{W}. \quad (2.13)$$

Since body forces have been neglected, the work done by the system on its surroundings is due to only surface forces. The work done by pressure \dot{W}_p is due to the product of the pressure force,

which acts normal to the boundary, and the velocity of the boundary in the normal direction. Hence,

$$\dot{W}_p = \oint_S p(\vec{v} \cdot \hat{n}) ds. \quad (2.14)$$

Similarly, the work done by viscous forces, \dot{W}_v , is due to the product of the viscous stresses and the velocity on the surface. Hence,

$$\dot{W}_v = - \oint_S \tau \cdot \vec{v} ds. \quad (2.15)$$

In the above equations, note that, by convention, work is defined as from the system to the surroundings.

The second way that energy can be transferred to the system across the surfaces is thermal diffusion via conduction, denoted by \dot{Q} . From Fourier's law, the heat diffused at any point in the fluid is

$$\vec{q} = -k\nabla T, \quad (2.16)$$

where k is the thermal conductivity of the fluid. Again, only the component of heat that is diffused normal to the surface of the control volume will actually enter it. Hence, the heat added to the system is

$$\dot{Q} = \oint_S k\nabla T \cdot \hat{n} ds, \quad (2.17)$$

again noting that heat transfer is defined as from the surroundings to the system.

From the work and heat transfer terms, we can now write an expression for conservation of energy

$$\frac{d}{dt} \int_{\Omega} \rho e d\vec{x} + \oint_S \rho e(\vec{v} \cdot \hat{n}) ds = \oint_S k\nabla T \cdot \hat{n} ds - \oint_S p(\vec{v} \cdot \hat{n}) ds + \oint_S (\tau \cdot \vec{v}) \cdot \hat{n} ds. \quad (2.18)$$

Similar to the momentum equation, this is often written more compactly as

$$\frac{d}{dt} \int_{\Omega} \rho e d\vec{x} + \oint_S [\rho e \vec{v} + p \vec{v} - \tau \cdot \vec{v} - k \nabla T] \cdot \hat{n} ds = 0. \quad (2.19)$$

2.1.4 Compact Integral Form

One might notice that the conservation of mass, momentum, and energy equations derived in the previous sections all have a similar form. They include a time derivative of the conserved variable integrated over the control volume, and a surface integral term of fluxes across the control volume surface. Commonly these equations are compacted into a vector of conserved quantities

$$\vec{w} = \begin{bmatrix} \rho \\ \rho \vec{v} \\ \rho e \end{bmatrix}, \quad (2.20)$$

a vector of inviscid fluxes

$$\vec{F}_{inv} = \begin{bmatrix} \rho \vec{v} \\ \rho \vec{v} \otimes \vec{v} + p \mathbf{I} \\ \rho e \vec{v} + p \vec{v} \end{bmatrix}, \quad (2.21)$$

and viscous fluxes

$$\vec{F}_{vis} = \begin{bmatrix} 0 \\ \tau \\ \tau \cdot \vec{v} - \vec{q} \end{bmatrix}. \quad (2.22)$$

This allows the integral form of the Navier-Stokes equations to be written compactly as

$$\frac{d}{dt} \int_{\Omega} \vec{w} d\vec{x} + \oint_S [\vec{F}_{inv} - \vec{F}_{vis}] \cdot \hat{n} ds = 0. \quad (2.23)$$

2.2 Divergence Form

Looking back at the previous section, we note that Equation 1.2 is a general conservation law for a finite control volume. In some contexts, specifically when using the finite-volume method that will be introduced later, this *integral form* of the governing equations is used. However, other approaches in CFD use a nearly equivalent *divergence form* of Equation 1.2. To derive this form, we rely on the divergence theorem, also known as Gauss theorem.

Theorem 2.2.1 — Divergence Theorem. The divergence theorem states that integrals of the following form are equivalent for a continuously differentiable vector field \vec{F}

$$\int_{\Omega} \nabla \cdot \vec{F} d\vec{x} = \oint_S \vec{F} \cdot \hat{n} ds, \quad (2.24)$$

which allows us to transform volume integrals into surface integrals, or the reverse.

2.2.1 Conservation of Mass

Starting from the integral form of conservation of mass and applying the divergence theorem to the surface term, we obtain

$$\frac{d}{dt} \int_{\Omega} \rho d\vec{x} + \int_{\Omega} \nabla \cdot (\rho \vec{v}) d\vec{x} = 0. \quad (2.25)$$

Since integration and differentiation commute, we can bring the time derivative inside of the first integral

$$\int_{\Omega} \frac{\partial \rho}{\partial t} d\vec{x} + \int_{\Omega} \nabla \cdot (\rho \vec{v}) d\vec{x} = 0. \quad (2.26)$$

and noticing that the bounds of both integrals are the same

$$\int_{\Omega} \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) \right) d\vec{x} = 0. \quad (2.27)$$

In order for this equation to be valid, other than in trivial cases, we require that the integrand be identically zero. Hence, conservation of mass in divergence form can be written as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0. \quad (2.28)$$

It is interesting to note that we have effectively converted a problem involving surface and volume integrals, into a differential form that requires computing derivatives.



A subtle difference between the two forms of conservation laws is that the integral form applies to control volumes and the divergence form applies at points. This will become important in choosing what form to use for CFD, and will be explored later.

2.2.2 Conservation of Momentum

Applying the same sets of operations to the integral form of the momentum equation, we can obtain its divergence form

$$\frac{\partial \rho \vec{v}}{\partial t} + \nabla \cdot [\rho \vec{v} \otimes \vec{v} - \sigma] = 0. \quad (2.29)$$

2.2.3 Conservation of Energy

Applying the same sets of operations to the integral form of the energy equation, we can obtain its divergence form

$$\frac{\partial \rho e}{\partial t} + \nabla \cdot [\rho e \vec{v} + p \vec{v} - \tau \cdot \vec{v} - k \nabla T] = 0. \quad (2.30)$$

2.2.4 Compact Divergence Form

Considering the compact integral form given in Equation 2.23, we notice that the divergence theorem can also be applied. Hence, a compact differential form of the Navier-Stokes equations can be obtained

$$\frac{\partial \vec{w}}{\partial t} + \nabla \cdot [\vec{F}_{inv} - \vec{F}_{vis}] = 0. \quad (2.31)$$

3. Simplified Systems

One might notice that the Navier-Stokes equations derived in the previous chapter are a complex system of coupled non-linear partial differential equations. This is not something that sounds particularly easy to solve! Hence, in CFD we often consider *simplified* systems of equations first, neglecting or decoupling some of the physical mechanisms that are involved in the full Navier-Stokes equations. This allows us to play with different ideas quickly and with relative ease. Then, once we understand how to solve different parts of these simplified equations, we will combine these ideas later to solve the full Navier-Stokes equations.

3.1 Euler Equations

The Euler equations, although they were actually derived prior to Navier-Stokes, can be obtained by simply neglecting viscous effects. Hence, we can ignore physical viscosity and thermal diffusion. While historically the Euler equations were the state-of-the-art in CFD, their lack of viscosity means they are unsuitable for predicting boundary layers. Nevertheless, they are still useful for predicting many flow phenomena, such as shockwaves.

In integral form the Euler equations are

$$\frac{d}{dt} \int_{\Omega} \rho d\vec{x} + \oint_S \rho \vec{v} \cdot \hat{n} ds = 0, \quad (3.1)$$

$$\frac{d}{dt} \int_{\Omega} \rho \vec{v} d\vec{x} + \oint_S [\rho \vec{v} \otimes \vec{v} + p \mathbf{I}] \cdot \hat{n} ds = 0, \quad (3.2)$$

$$\frac{d}{dt} \int_{\Omega} \rho e d\vec{x} + \oint_S [\rho e \vec{v} + p \vec{v}] \cdot \hat{n} ds = 0, \quad (3.3)$$

for conservation of mass, momentum, and energy, respectively. Similarly, in divergence form the Euler equations are

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0, \quad (3.4)$$

$$\frac{\partial \rho \vec{v}}{\partial t} + \nabla \cdot [\rho \vec{v} \otimes \vec{v} + p \mathbf{I}] = 0, \quad (3.5)$$

$$\frac{\partial \rho e}{\partial t} + \nabla \cdot [\rho e \vec{v} + p \vec{v}] = 0, \quad (3.6)$$

for conservation of mass, momentum, and energy.

3.2 Linear Advection

Even if we consider the Euler equations, we notice that they are still relatively complex and difficult to solve. In what follows, we will use a set of thought experiments to generate a set of three much simpler equations that will be a starting point for our initial exploration of CFD. To start, we will derive the so-called linear advection equation. We begin our thought experiment by considering a fluid flow that has uniform velocity throughout the domain. Furthermore, we will decouple conservation of mass from the other two conservation laws.

Now we can imagine that our fluid flow, with constant velocity everywhere such that $\vec{v}(\vec{x}, t) = \vec{\alpha}$, has some blob of fluid that is dense relative to the rest of the fluid around it. For example, it could be slightly colder, increasing its density. What should happen to this blob of fluid over time? Well, since the fluid is all moving at the same velocity $\vec{\alpha}$, we would expect the blob of dense fluid to simply move along with the rest of the flow and, hence, the blob should move at velocity $\vec{\alpha}$, as shown in Figure 3.1.

Mathematically, this yields the following integral and differential forms for the linear advection equation by using conservation of mass and replacing the velocity by a constant velocity field $\vec{\alpha}$

and the density by a general scalar u we obtain,

$$\frac{d}{dt} \int_{\Omega} u d\vec{x} + \oint_S (\vec{\alpha} u) \cdot \hat{n} ds = 0, \quad (3.7)$$

and

$$\frac{\partial u}{\partial t} + \nabla \cdot (\vec{\alpha} u) = 0. \quad (3.8)$$

Furthermore, if we restrict ourselves to one-dimensional problems, we obtain the following integral and differential forms for linear advection

$$\frac{d}{dt} \int_{\Omega} u dx + \alpha \oint_S u \cdot \hat{n} ds = 0, \quad (3.9)$$

and

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0. \quad (3.10)$$

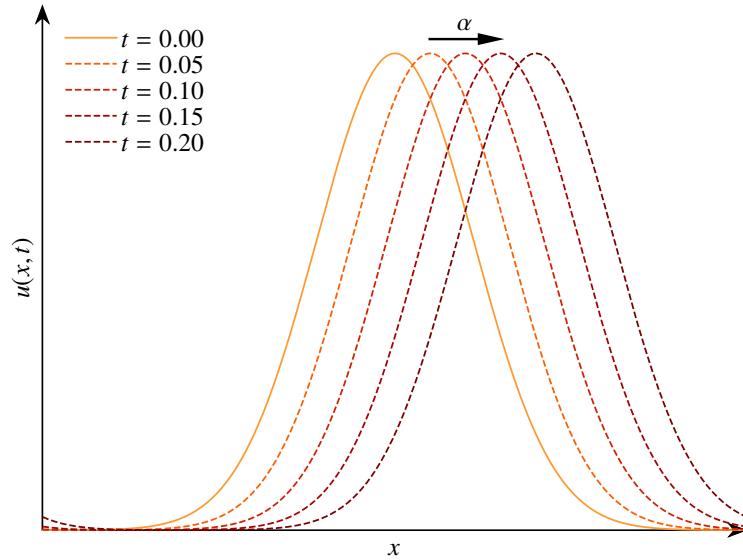


Figure 3.1: Evolution of a gaussian bump $u(x, 0) = \exp \left[-40 \left(x - \frac{1}{2} \right)^2 \right]$ using the advection equation on $x \in [0, 1]$ with velocity $\alpha = 1$, and periodic boundary conditions.

3.3 Burgers Equation

Our second simplified system, known as Burgers equation, is useful as a simplified model for compressible flow features such as shocks and expansion fans. To derive Burgers equation, we start with the momentum equation, decoupled from conservation of mass and conservation of energy. Then, neglecting the effects of viscosity and pressure, we replace the momentum with an arbitrary conserved variable u , and restrict ourselves to one physical dimension.

This yields the following integral and differential forms of the Burgers equation

$$\frac{d}{dt} \int_{\Omega} u dx + \frac{1}{2} \oint_S u^2 \cdot \hat{n} ds = 0, \quad (3.11)$$

and

$$\frac{\partial u}{\partial t} + \frac{1}{2} \frac{\partial u^2}{\partial x} = 0, \quad (3.12)$$

noting that the factor of one half is added to the spatial term by convention.

If we consider the divergence form of Burgers' equation, applying the chain rule to the spatial derivative operator yields

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0. \quad (3.13)$$

We note that this looks remarkably similar to the divergence form of the linear advection equation, also in one dimension. However, the advection velocity α , which appears in front of the spatial derivative for linear advection, has instead been replaced by u , the value of the solution. Hence, Burgers equation has similar behaviour to linear advection, except the velocity at any point in space is *equal* to the value of the solution at that point, rather than being a constant value throughout the domain. An example of this is shown in Figure 3.2, where the solution at any point moves at a speed equal to its value. This causes the bump to deform, with the forward-moving peak catching up with the slower moving fluid in front of it. This causes the formation of a discontinuity, or shockwave, in the solution on the right hand side of the domain.

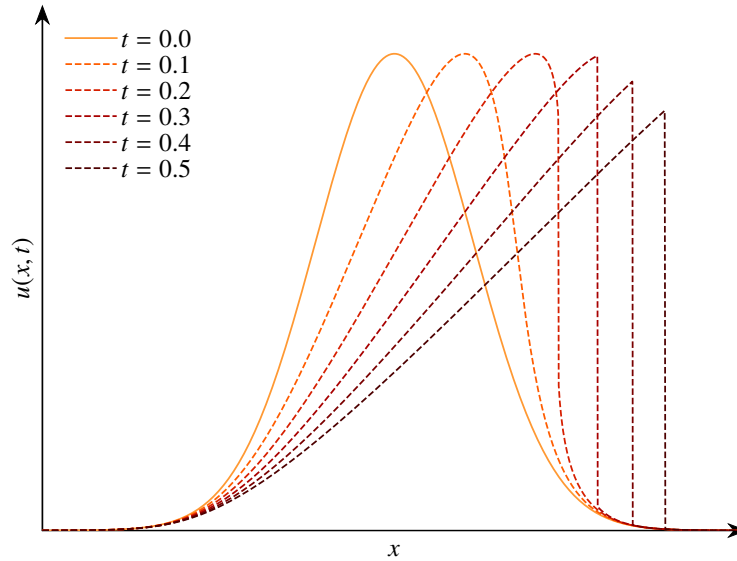


Figure 3.2: Evolution of a gaussian bump $u(x, 0) = \exp \left[-40 \left(x - \frac{1}{2} \right)^2 \right]$ using Burgers equation on $x \in [0, 1]$.

3.4 Linear Diffusion

Our third and final simplified equation, known as linear diffusion, starts again from a simple thought experiment. First, we will consider only the energy equation decoupled from conservation of mass and momentum. We will now imagine that we have a stationary fluid with zero velocity everywhere in the domain. Similar to linear advection, we will consider a flow with some blob of fluid with

more energy than the fluid around it. Since all of the fluid is stationary, this extra energy must come in the form of heat. As time goes on, we would expect that this local region of hot fluid would diffuse some of its heat over time to the cold fluid that is adjacent to it. Hence, over time an initially concentrated blob of heat would spread out, until eventually all of the fluid is at the same temperature.

Mathematically, the equation describing this can be obtained by taking $\beta = k/\rho c_v$ as a constant scalar and replacing e with a generic scalar u . We can then rewrite the energy equation in multiple dimensions as

$$\frac{d}{dt} \int_{\Omega} u d\vec{x} + \oint_S [-\beta \nabla u] \cdot \hat{n} ds = 0. \quad (3.14)$$

and

$$\frac{\partial u}{\partial t} - \nabla \cdot (\beta \nabla u) = 0, \quad (3.15)$$

in integral and divergence form, respectively. Furthermore, if we restrict ourselves to one dimension we obtain

$$\frac{d}{dt} \int_{\Omega} u dx + \oint_S \left[-\beta \frac{\partial u}{\partial x} \right] \cdot \hat{n} ds = 0. \quad (3.16)$$

and

$$\frac{\partial u}{\partial t} - \beta \frac{\partial^2 u}{\partial x^2} = 0. \quad (3.17)$$

At this point, it is worth noting that the form of the linear diffusion equation is similar to the linear advection equation, except we are taking the second derivative rather than the first. An example of the evolution of the diffusion equation is shown in Figure 3.3, demonstrated that as the solution evolves the concentrated energy is spread out to the surrounding fluid.

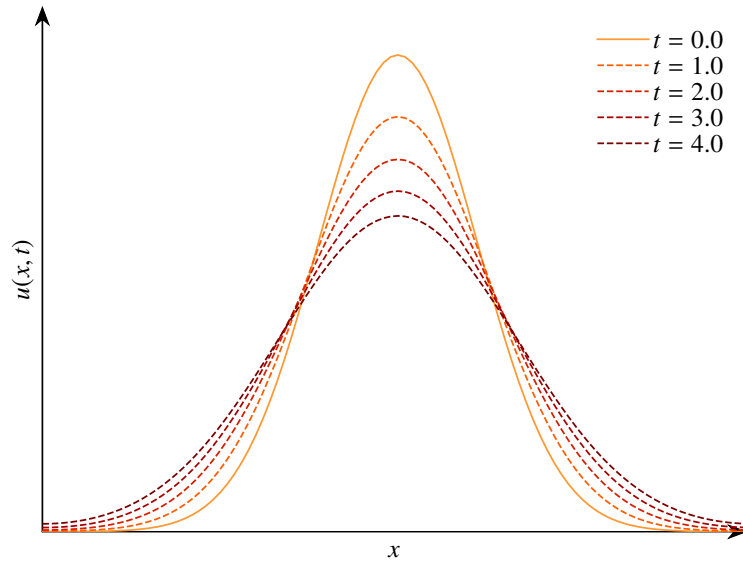


Figure 3.3: Evolution of a gaussian bump $u(x,0) = \exp \left[-40 \left(x - \frac{1}{2} \right)^2 \right]$ using the linear diffusion equation on $x \in [0, 1]$.

3.5 PDE Classification

In the previous sections, we have introduced several different partial differential equations. From the context of the Navier-Stokes equations, these include conservation of mass, momentum, and energy, which we have then simplified into the Euler, linear advection, Burgers, and linear diffusion equations. As we will see in later sections, not all numerical approaches work well for all partial differential equations, and it is often useful to classify them based on their properties and behaviour.

3.5.1 First Order Equations

First order partial differential equations take the form

$$A \frac{\partial u}{\partial x} + B \frac{\partial u}{\partial y} = F(x, y, u). \quad (3.18)$$

Note that the x and y dimensions here need not be only space, this is just a general form. Hence, the linear advection equation is also of this form, since both of its derivatives are first order in both space and time. These are always *hyperbolic* in nature, and as a result, they exhibit wavelike solutions. This means that information travels in a particular defined direction, as was demonstrated for the linear advection and Burgers equations.

3.5.2 Second Order Equations

Second order partial differential equations take the form

$$A \frac{\partial^2 u}{\partial x^2} + B \frac{\partial^2 u}{\partial x \partial y} + C \frac{\partial^2 u}{\partial y^2} = F(x, y, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}). \quad (3.19)$$

Depending on the values of A , B , and C , these type of equations will exhibit different behaviour.

$$B^2 - 4AC > 0$$

These are also hyperbolic in nature, and exhibit wave-like solutions.

$$B^2 - 4AC = 0$$

These are parabolic in nature, and are typically transient diffusion processes.

$$B^2 - 4AC < 0$$

These are elliptic in nature, and are typically steady-state diffusion processes.

If we look at linear advection, it is first order and, therefore hyperbolic. If we look at the linear diffusion equation, we find that it is parabolic. Hence, we expect that the numerical behaviour of these two different problems will be quite different. A few other examples include the classical wave equation

$$\frac{\partial^2 u}{\partial t^2} - \alpha \frac{\partial^2 u}{\partial x^2} = 0, \quad (3.20)$$

which is hyperbolic and should have similar behaviour to the linear advection equation. Also, a steady-state two dimensional diffusion problem has the form

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad (3.21)$$

which is elliptic, and will typically require a different solution strategy.

The above is a relatively simple classification, but when confronted with a new type of partial differential equation, it is very useful to identify its classification to see if its behaviour is similar to another well-known system. In addition, in many applications the A , B , and C coefficients can be a function of space, time, or a non-linear function of the solution. Hence, the behaviour of the system can change from one type to another as the solution evolves. In this case, a particularly robust numerical approach is required.

4. Turbulence

One of the most challenging aspects of CFD is the prevalence of turbulent flows. While it may not be immediately apparent from the Navier-Stokes equations defined earlier, they can encode solutions with chaotic, unsteady, three-dimensional flow features. In this section, we will discuss what is meant by turbulent flow, the nature of chaos, how it arises in the governing equations, and consequences for how we handle turbulent flows in CFD. Then, we will introduce an approach for approximating the effects of turbulence, and several popular models for doing so. Some other useful references for this section include [4, 15].

4.1 Turbulence Theory

The fundamental characteristic of turbulence is that it is *chaotic*. Hence, it encodes unsteady three-dimensional fluid flow with chaotic fluctuations in velocity, density, and pressure. These fluctuations typically exist over a wide range of length and time scales. Another important feature of turbulence, and a fundamental property of chaotic systems, is that it is highly sensitive to initial conditions. Furthermore, it is well known that chaotic behaviour only arises in *non-linear systems*. In our exploration of turbulence we will first start with a simple chaotic system, and then we will extrapolate some of the properties of these systems to the types of behaviour we observe in fluid flows.

4.1.1 Introduction to Chaos

As an introduction to chaos, we will consider the relatively simple logistic map, which is built from the non-linear logistic function [21]

$$x_{n+1} = ax_n(1 - x_n), \quad (4.1)$$

where x_n is the current value, x_{n+1} is the next value, and a is a scalar that governs the behaviour of the system. A relatively simple analogy/application of the logistic equation is the growth/decay of a population of animals. In this case x_n would be the current population, x_{n+1} is next years population, and a is responsible for controlling the birth/death rate of the animals in any given year based on the current population. The first thing to notice is that this equation is incredibly simple, yet under certain conditions it encodes infinitely complex chaotic behaviour, and the behaviour of the system is governed by the choice of a .

$$a \leq 3$$

In this case, regardless of the initial value x_0 , the solution converges to one of two fixed points, either $x = 0$, or $x = (a - 1)/a$. In the context of a population prediction, this means that either all of the animals die, or the population converges to a constant value.

$$3 < a \leq 1 + \sqrt{6}$$

In this case, the system rapidly flips back and forth between two different values, referred to as a *two-cycle*. Regardless of the initial condition, the system will converge to one of these values, and then flip alternating back and forth between them. The transition from the initial fixed-point solution to this two-cycle is referred to as a *bifurcation*. From a population perspective, this means alternating years of famine, where many animals die, and feast, where many animals are born.

$$1 + \sqrt{6} < a \leq 3.54\dots$$

In this case, a second bifurcation occurs, and the solution oscillates between four possible solutions. This is referred to as a *period doubling*. As a consequence, the population will be a more complex cycle of feast and famine.

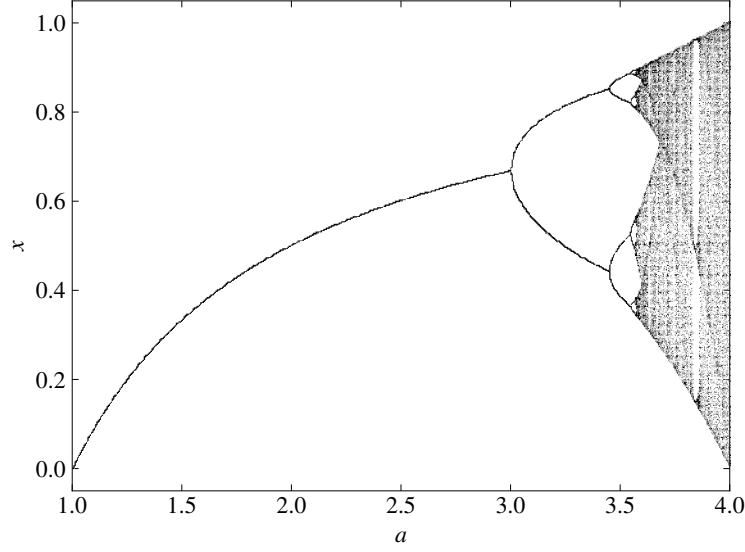
$$\leq 3.54\dots < a \leq 3.57\dots$$

Beyond $a = 3.54\dots$, additional bifurcations and period doublings occur, with the population now oscillating between an ever-growing number of possible values. By $a = 3.57\dots$, an infinite number of bifurcations occurs.

$$3.57\dots < a$$

Beyond $a = 3.57\dots$, the entire concept of a cycle breaks down, and all possible solution can be found within the range. The solution traverses chaotically and little structure is observed in its behaviour.

The complete behaviour of Equation 4.1 can be observed in Figure 4.1 for a range of values of a in the range $a \in [1, 4]$. An interesting thought experiment is to consider what happens to our system when we start with some initial guess x_0 , and some perturbed initial guess $x_0 + \epsilon$, where ϵ is a small number. In the case of small values of a both of these solution converge to the same fixed point. However, as a gets larger, this predictability starts to break down. Eventually, with a large value of a , and given enough iterations, the two solutions will diverge from one another, and it will be impossible to discern that they had almost the exact same initial condition. Hence, with certain choices of a the logistic equations is chaotic, resulting in complex behaviour and extreme sensitivity to initial conditions. Furthermore, this implies that there is an *irreversible* nature to chaotic systems, that is, given a final solution, it is not possible to determine with any certainty what the initial condition was, because very similar initial conditions yield completely different final solutions.

Figure 4.1: Logistic map for $a \in [1, 4]$ 

Check out the Chaos Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

4.1.2 Chaos and Navier-Stokes

In the above, we have demonstrated a couple of properties of chaos. It arises in non-linear systems, and it is generally irreversible due to an extreme sensitivity to initial conditions. For simplicity, we will explore the connection between the Navier-Stokes equations and chaos, in the form of turbulence, via the incompressible Navier-Stokes equations. This can also be readily extended to the compressible Navier-Stokes equations at the readers' initiative.

In the case of incompressible flow, the Navier-Stokes equations reduce to the continuity equation

$$\nabla \cdot \vec{v} = 0, \quad (4.2)$$

and the momentum equation

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla) \vec{v} - \nu \nabla^2 \vec{u} = 0, \quad (4.3)$$

where ν is the kinematic viscosity. Starting with conservation of mass, we note that this is a linear equation and, hence, it cannot be responsible for the chaotic nature of turbulence, which requires a non-linear system as previously noted. This allows us to quickly narrow in on the momentum equation as the responsible part of the system. In this case, we note that the non-linearity arises in the $(\vec{v} \cdot \nabla) \vec{v}$ term, which is quadratic in the velocity components. To better understand this, we will consider two limiting cases of inviscid and high viscosity flows.

Starting with the inviscid case, we can neglect the effects of viscosity, yielding the following simplification of the momentum equation

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla) \vec{v} = 0, \quad (4.4)$$

which is the momentum component of the incompressible Euler equations. This is obviously still a non-linear equation, since we have not eliminated the convection term. We can now ask ourselves

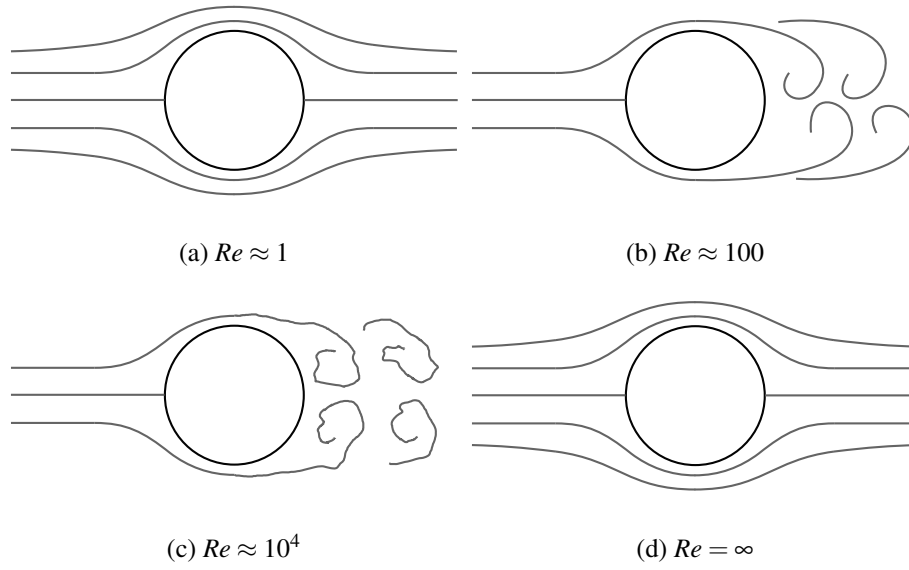


Figure 4.2: Incompressible flow regimes at different Reynolds numbers

if this inviscid flow can permit chaotic solutions. Recall that our second requirement for a chaotic system is that it should be irreversible. In order to explore this we will reverse this momentum equation in time by setting $t = -t$ and $\vec{v} = -\vec{v}$, which is equivalent to stopping a solution and then rewinding it in time. Applying these modifications to the forward in time system yields

$$\frac{\partial(-\vec{v})}{\partial(-t)} + ((-\vec{v}) \cdot \nabla)(-\vec{v}) = 0, \quad (4.5)$$

which simply gives us back our original equation

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla) \vec{v} = 0. \quad (4.6)$$

Hence, the incompressible Euler equations are completely time reversible and, hence, cannot permit chaotic solutions. Therefore, we need viscosity to enable turbulence.

R Careful consideration should be used when extrapolating these conclusions to the compressible Euler equations. In that case, irreversibility can arise across shockwaves, which increase entropy. Hence, turbulence can be observed with the compressible Euler equations in some cases.

Now, let's explore the alternative of a purely viscous flow, which is the limit that viscosity goes to extremely large values. This yields the following simplification of the momentum equation

$$\frac{\partial \vec{v}}{\partial t} - \nu \nabla^2 \vec{u} = 0, \quad (4.7)$$

which is a linear equation in velocity. Hence, in the limit of large viscosities, the system of equations becomes linear and can no longer permit chaotic solutions. From the above discussion, it is clear that the momentum equations in the incompressible Navier-Stokes equation are non-linear with respect to the velocity. Furthermore, viscosity is key in determining whether chaotic solutions exist. If there is not enough, the system becomes reversible. In contrast, if there is too much, the system becomes linear. The relative strength of the viscosity is given by the Reynolds number

$$Re = \frac{UL}{\nu}, \quad (4.8)$$

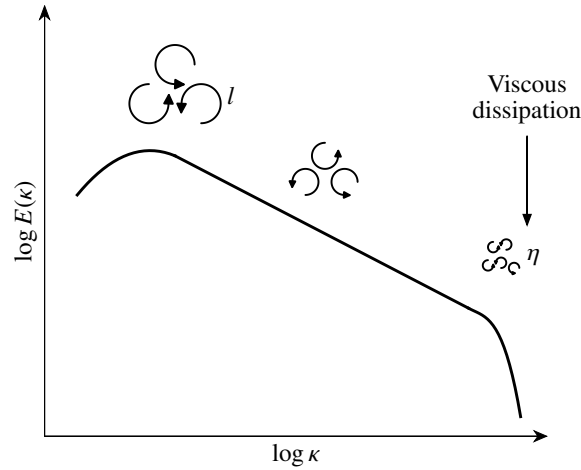


Figure 4.3: Turbulent kinetic energy cascade

where U and L are the velocity and length scales of the flow, and ν is the viscosity. Examples of different Reynolds numbers can be observed in Figure 4.2. We can note here the similarity of dependence of turbulence on the Reynolds number to the behaviour of the logistic equation on a . Typically, turbulent flow is observed for Reynolds numbers of $Re \gtrsim \mathcal{O}(10^3)$. It is important to note that this encompasses the vast majority of flows of engineering interest. Hence, scientists and engineers must routinely deal with turbulence and its chaotic nature.

4.1.3 The Energy Cascade

A common misconception is that turbulence is *random* in nature, which would imply that it has no pattern and is inherently unpredictable in a deterministic sense. However, the very fact that one can derive the Navier-Stokes equations, which are a deterministic model for fluid flow, implies that turbulence is not a random process. Hence, in the lack of detailed measurement techniques, turbulence is often *modelled* as random process, but its evolution is truly deterministic in nature. Furthermore, turbulent flows clearly have coherent structures, such as turbulent vortices, that demonstrate an underlying structure within the chaos.

R We will revisit how to model turbulence stochastically when we introduce the Reynolds Averaged Navier-Stokes (RANS) equations and turbulence models.

One of the most powerful observations of turbulent flow has to do with the distribution of kinetic energy across different physical scales. When considering turbulent flows, there is typically a wide distribution of length scales and time scale with the largest vortices have a size similar to the physical length scale of the problem of interest, such as an airfoils chord length or the size of a bluff body such as a cylinder. In contrast, the smallest physical scales can be much smaller than the physical length scale of the problem, and there is then a wide range of scales between these two extremes. A fundamental observation related to this is that these turbulent structures, over time, tend to break up into multiple smaller structures. A consequence of this is that kinetic energy gradually gets transferred from large-scale structures down to smaller-scale structures and so on, referred to as the *turbulent kinetic energy cascade*, which can be observed in Figure 4.3.

To better understand this, we will define l as the length scale of the largest scale vortices, and u as their velocity scale. Similarly, we will define η as the length scale of the smallest scale vortices and v as their velocity scale. For the large scale vortical structures, they will complete a revolution,

or turnover time, in approximately

$$t \sim \frac{l}{u}. \quad (4.9)$$

In experiments, it is observed that a turbulent structure tends to break up within a small number of eddy turnover times, regardless of its size. Hence, the rate at which kinetic energy gets transferred from the largest scales is

$$\Pi \sim \frac{u^2}{t}, \quad (4.10)$$

which expands to

$$\Pi \sim \frac{u^3}{l}. \quad (4.11)$$

Hence, if we know the approximate size and velocity of the largest scale turbulent structures, we can estimate the rate at which they transfer energy down to the smaller scales.

Similarly, if we consider the small scale structures, it can be shown via the Navier-Stokes equations that they dissipate their energy via viscous effects into heat at a rate of [4]

$$\varepsilon \sim \nu S_{ij} S_{ij} \sim \nu \frac{v^2}{\eta^2}, \quad (4.12)$$

where ε is the rate of kinetic energy dissipation into heat and S_{ij} is the strain rate tensor. Hence, we can also approximate the rate at which kinetic energy ultimately gets turned into heat.

If we consider a quasi steady system, where the rate at which kinetic energy gets introduced to the flow at the large scales has stabilized, then it must balance with the rate at which kinetic energy gets dissipated by the smallest scales. Hence, under these conditions

$$\Pi = \varepsilon, \quad (4.13)$$

which yields

$$\frac{u^3}{l} \sim \nu \frac{v^2}{\eta^2}. \quad (4.14)$$

Considering again the smallest scale structures, we would expect the dissipation due to viscous effects to start to dominate when they are about the same strength as the inertial effects. Hence, at the smallest scales, the kinetic energy is expected to diffuse into heat when

$$Re = \frac{\nu \eta}{\nu} \approx 1. \quad (4.15)$$

In conjunction with Equation 4.14, this yields

$$\frac{l}{\eta} \sim Re^{3/4}. \quad (4.16)$$

Hence, if we know the Reynolds number of the flow, we can estimate the approximate size of the largest scales relative to the smallest scales, referred to as the scale separation. A similar procedure for the velocity components yields

$$\frac{v}{u} \sim Re^{-1/4}, \quad (4.17)$$

which can be used to estimate the relative difference in the velocity magnitudes at the largest and smallest scales as well.

Now, let's consider what this means in the context of CFD. If we have an object that we want to simulate flow over, the domain size needs to be at least as large as the object, which would be l . Furthermore, if we want to resolve the smallest scale turbulent structures, then the grid resolution must be approximately the same size as the structures, η . Furthermore, this mesh must be three dimensional. Hence, we can estimate the total number of grid points that would be required from

$$N \sim \left(Re^{3/4}\right)^3, \quad (4.18)$$

which expands to

$$N \sim Re^{9/4}, \quad (4.19)$$

where N is the total number of grid points. Hence, as the Reynolds number goes up, the required number of grid points scales extremely rapidly with the Reynolds number. For practical Reynolds numbers above $Re \sim 10^5$, this rapidly becomes intractable on modern high-performance computers. Therefore, if we want to simulate turbulent flows above $Re \sim 10^5$, we simply cannot resolve the large-scale structures and the small-scale structures in their entirety at the same time. The most common solution to this, which will be explored in the following section, is to resolve only the large scale structures and introduce a *model* for the effect the small scale structures will have on the large scale structures. This model will not be able to exactly mimic the effects of the small scale structures and, hence, it will introduce some error into our solution of the largest scales.

4.2 Reynolds Averaging

Now that we have briefly introduced chaos and how it relates to the Navier-Stokes equations, it is clear that the study of turbulent flows is indeed challenging. Small changes in the initial conditions may substantially deviate the results. Furthermore, we have demonstrated that the scale separation between the largest and smallest turbulent structures rapidly becomes intractable as the Reynolds number increases. Hence, we commonly require the use of statistical tools to understand its behaviour, which includes averaging operations. One of the most widely used, known as *Reynolds decomposition*, was introduced by Osborne Reynolds [17] in 1895. This concept relies on representing any flow property $u(\vec{x}, t)$ as the sum of its mean and fluctuating components, such that

$$u(x, t) = \overline{u(\vec{x})} + u(\vec{x}, t)', \quad (4.20)$$

where $\bar{\cdot}$ denotes the averaged term and $'$ the fluctuation. Turbulent flows can be considered *stationary* if, after averaging, they do not vary in time. Even though the instantaneous fluctuating components will vary, we gain some reproducibility by considering the mean quantities instead. An example could be a wing of an airplane flying at cruise. It is clear that if we measure the velocity at a given point where the flow is turbulent, we will observe fluctuations in the results. However, if these results are *time-averaged*, we will expect to obtain the same value should the experiment be repeated. This type of averaging is defined in an integral sense as

$$u_T(\vec{x}) = \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \int_{t_0}^{t_0 + \tau} u(\vec{x}, t) dt, \quad (4.21)$$

where τ is the length of the averaging. For large values of τ , u_T is independent of the initial conditions. A schematic representation can be observed in Figure 4.4. We observe a constant value given the stationary characteristic of the flow.

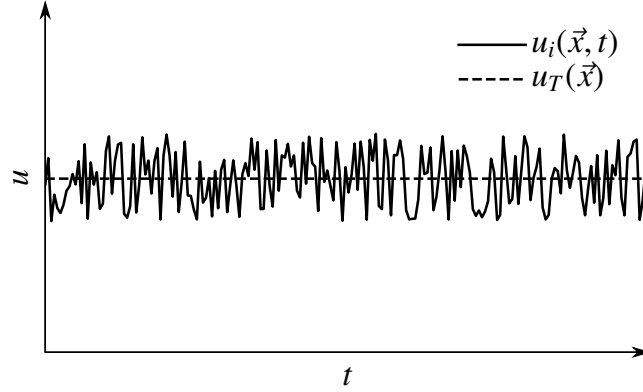


Figure 4.4: Time averaging of a stationary flow quantity $u(\vec{x}, t)$.

Homogeneous flows are those whose properties do not vary in any direction. In this case, a *spatial averaging* procedure may be convenient

$$u_{\Omega}(t) = \lim_{\Omega \rightarrow \infty} \frac{1}{\Omega} \int_{\Omega} u(\vec{x}, t) dV, \quad (4.22)$$

where Ω is the volume of the domain. Finally, *ensemble averaging* can be used to obtain mean values using N identical experiments, even if initial and boundary conditions include infinitesimal perturbations. In this type of averaging, the variation on both the spatial and temporal coordinates is maintained. For N experiments, where N is large, we can calculate

$$u_N(\vec{x}, t) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=1}^N u_i(\vec{x}, t), \quad (4.23)$$

where $u_i(\vec{x}, t)$ is the result obtained in the i -th experiment. An example of ensemble averaging can be seen in Figure 4.5 for non-stationary flows, where we discover a smooth sinusoidal behaviour after N experiments have been performed.

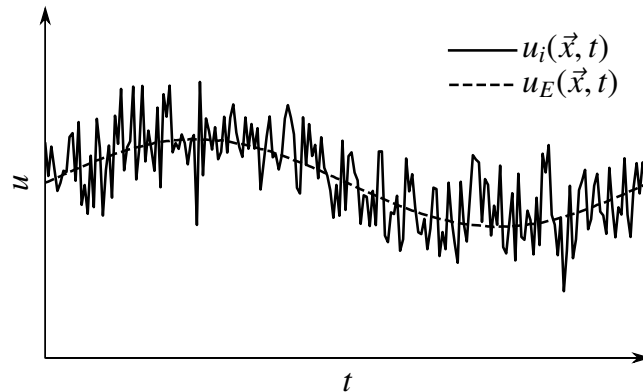


Figure 4.5: Ensemble averaging results of a non-stationary flow quantity $u(\vec{x}, t)$.

In the following section, we will derive the Reynolds Averaged Navier-Stokes equations using the time-averaging operation. Before we dive into the derivation, we will discuss some Reynolds Averaging properties. For a complete explanation of these properties, refer to Wilcox [25].

Properties of Reynolds Averaging

Consider a statistically-steady state flow scenario, two flow quantities u and v , and constants α and β .

- Since the time-averaging process deals with definite integrals, it is a linear operation. Hence,

$$\overline{\alpha u + \beta v} = \alpha \bar{u} + \beta \bar{v}. \quad (4.24)$$

- The time averaging of integrals and derivatives commute

$$\overline{\int u dy} = \int \bar{u} dy, \quad \overline{\frac{\partial u}{\partial t}} = \frac{\partial \bar{u}}{\partial t}. \quad (4.25)$$

- Any time-averaged fluctuation vanishes, i.e.

$$\bar{u'} = 0. \quad (4.26)$$

- The average of the product of two instantaneous quantities is given by

$$\overline{uv} = \bar{u}\bar{v} + \overline{u'v'}. \quad (4.27)$$

- From the aforementioned properties, it can also be shown that

$$\bar{\bar{u}} = \bar{u}, \quad \overline{u'\bar{v}} = 0. \quad (4.28)$$

4.2.1 The Reynolds Averaged Navier-Stokes Equations

The Reynolds-Averaged Navier-Stokes (RANS) equations are especially useful for analysis of time-marching numerical methods involving statistically steady flow problems. These equations have been widely used in the aerospace industry for decades. The idea is to split each of the flow variables into mean and fluctuating components, and then perform time averaging on the governing equations. As a consequence, high-frequency information is removed from the flow via the averaging procedure. The resulting conservation laws describe only the evolution of the mean flow quantities, which are typically sufficient to determine terms of engineering interest, such as the lift and drag coefficients.

In this section, we will derive the RANS equations for incompressible flow, and later for the compressible case. We will then discuss the appearance of the Reynolds stresses as a consequence of time-averaging the non-linear convective terms from the Navier-Stokes equations.

Incompressible flow

Consider the incompressible mass conservation law

$$\nabla \cdot \vec{v} = 0, \quad (4.29)$$

which we rewrite using tensor notation

$$\frac{\partial v_i}{\partial x_i} = 0, \quad (4.30)$$

where index i indicates summation over all considered directions. We decompose the velocity variable as the sum of mean and fluctuation components

$$\overline{\frac{\partial}{\partial x_i} (\bar{v}_i + v'_i)} = 0. \quad (4.31)$$

Since the time-averaging procedure commutes with derivative operators (Equation 4.25),

$$\frac{\partial}{\partial x_i} \overline{(\bar{v}_i + v'_i)} = 0. \quad (4.32)$$

Using Equations 4.24 and 4.26,

$$\frac{\partial \bar{v}_i}{\partial x_i} = 0, \quad (4.33)$$

where the fluctuating term has vanished. This means that the divergence of each of the terms in Equation 4.32 must equal zero. Hence

$$\frac{\partial v'_i}{\partial x_i} = 0. \quad (4.34)$$

Next, we consider the momentum equation. Similarly, we split the variables using Reynolds decomposition and then time-average both sides of the equation. The momentum equations can be written using tensor notation

$$\rho \frac{\partial v_i}{\partial t} + \rho v_j \frac{\partial v_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \mu \frac{\partial^2 v_i}{\partial x_j^2}, \quad (4.35)$$

where μ is the dynamic viscosity and $\rho = \bar{\rho}$ is constant due to the incompressible condition. We will simplify the derivation beforehand by considering

$$\frac{\partial (v_j v_i)}{\partial x_j} = v_j \frac{\partial v_i}{\partial x_j} + v_i \frac{\partial v_j}{\partial x_j}, \quad (4.36)$$

where the second term $\partial v_j / \partial x_j$ vanishes according to continuity. Hence, we write

$$\frac{\partial (v_j v_i)}{\partial x_j} = v_j \frac{\partial v_i}{\partial x_j}, \quad (4.37)$$

which we use to replace the convective term in Equation 4.35, such that

$$\rho \frac{\partial v_i}{\partial t} + \rho \frac{\partial v_j v_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \mu \frac{\partial^2 v_i}{\partial x_j^2}. \quad (4.38)$$

Decomposing the flow variables and time-averaging the resulting equation, we have

$$\rho \frac{\partial (\bar{v}_i + v'_i)}{\partial t} + \rho \frac{\partial (\bar{v}_j + v'_j) (\bar{v}_i + v'_i)}{\partial x_j} = -\frac{\partial (\bar{p} + p')}{\partial x_i} + \mu \frac{\partial^2 (\bar{v}_i + v'_i)}{\partial x_j^2}. \quad (4.39)$$

By first resolving multiplications and then applying the Reynolds averaging properties, we obtain the RANS equations for incompressible flow

$$\rho \frac{\partial \bar{v}_i}{\partial t} + \rho \frac{\partial}{\partial x_j} (\bar{v}_j \bar{v}_i + \overline{v'_j v'_i}) = -\frac{\partial \bar{p}}{\partial x_i} + \mu \frac{\partial^2 \bar{v}_i}{\partial x_j^2}. \quad (4.40)$$

Recall the stress tensor τ_{ij} from Chapter 4, which we now rewrite in tensor notation

$$\tau_{ij} = 2\mu S_{ij}, \quad (4.41)$$

where S_{ij} is the strain-rate tensor

$$S_{ij} = \frac{1}{2} \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right). \quad (4.42)$$

We may now rewrite the RANS momentum equation in terms of the stresses and rearrange some terms, such that

$$\rho \frac{\partial \bar{v}_i}{\partial t} + \rho \frac{\partial}{\partial x_j} (\bar{v}_j \bar{v}_i) = -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j} (\bar{\tau}_{ji} - \rho \overline{v'_j v'_i}), \quad (4.43)$$

where $\rho \overline{v'_j v'_i}$ is known as the *Reynolds stress*, which we can define as the *Reynolds stress tensor* $\rho \hat{\tau}_{ij}$, such that

$$\hat{\tau}_{ij} = -\overline{v'_i v'_j}. \quad (4.44)$$

Note that the Reynolds stress tensor is symmetric since $\hat{\tau}_{ij} = \hat{\tau}_{ji}$. This means that three diagonal components and three off-diagonal components make a total of six independent variables only in $\hat{\tau}_{ij}$. This number of unknowns rises to ten if we consider the conserved properties of three-dimensional flow, with density as well as one momentum equation in each direction. Clearly, we have an underdetermined system and require additional equations. Later in this chapter, we derive an equation for the Reynolds stresses and discuss some of the consequences that arise from the averaging of the Navier-Stokes equations. We now look at the RANS equations for compressible flow.

Compressible flow

In the previous section, we derived the incompressible form of the RANS equations, where the density variations are negligible, i.e. $\rho = \bar{\rho}$. In the case of compressible flow, these variations must be taken into account. However, the equations can become very lengthy and complicated if the conventional averaging procedure is used. To visualize this, let us decompose the mass conservation law using the conventional Reynolds decomposition

$$\frac{\partial}{\partial t} (\bar{\rho} + \rho') + \frac{\partial}{\partial x_i} [(\bar{\rho} + \rho') (\bar{v}_i + v'_i)] = 0, \quad (4.45)$$

which will introduce correlations involving density fluctuations and may complicate the turbulence modelling, i.e.

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial}{\partial x_i} (\bar{\rho} \bar{v}_i + \overline{\rho' v'_i}) = 0. \quad (4.46)$$

By performing a density-weighted time-averaging, introduced by Alexandre Favre in 1969, the equations become simpler and the presence of ρ' can be avoided. It is defined by

$$\tilde{u} = \frac{1}{\bar{\rho}} \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{v=1}^{v=n} (\rho u)^{(v)} = \frac{\overline{\rho u}}{\bar{\rho}}, \quad (4.47)$$

where $\bar{\cdot}$ represents the conventional time-averaging. Similar to Reynolds decomposition, we consider the flow properties to be the sum of a mean and fluctuating component

$$u = \tilde{u} + u'', \quad (4.48)$$

where \tilde{u} is the *Favre mean* and u'' is the *Favre fluctuation*. Some properties of the Favre averaging are

$$\tilde{\tilde{u}} = \tilde{u}, \quad (4.49)$$

$$\overline{\rho u''} = 0, \quad (4.50)$$

$$\overline{u''} = -\overline{\rho' u'} / \bar{\rho}, \quad (4.51)$$

$$\overline{\rho u v} = \bar{\rho} \tilde{u} \tilde{v} + \overline{\rho u'' v''}. \quad (4.52)$$

We note that ρ is the instantaneous density and that the Favre average differs from the Reynolds averaging properties. For instance, $\overline{u'} = 0$, whereas $\overline{u''} \neq 0$. In addition, the conventional Reynolds and Favre averaging are related by

$$\tilde{u} = \bar{u} + \frac{\overline{\rho' u'}}{\bar{\rho}}, \quad (4.53)$$

$$u'' = u' - \frac{\rho' u'}{\bar{\rho}}. \quad (4.54)$$

We now derive the compressible RANS equations, also known as Favre-Averaged Navier-Stokes Equations (FANS). Starting with conservation of mass, by performing time averaging, we have

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial \bar{\rho} v_i}{\partial x_i} = 0. \quad (4.55)$$

Since the time-averaging procedure commutes with the derivatives,

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial \bar{\rho} v_i}{\partial x_i} = 0. \quad (4.56)$$

The first term can then be Favre decomposed and time averaged

$$\frac{\partial}{\partial t} (\bar{\rho} + \rho'') = \frac{\partial \bar{\rho}}{\partial t}, \quad (4.57)$$

and the proof is left as a simple exercise to the reader. The second term $\bar{\rho} v_i$ can be computed using the Favre definition in Equation 4.47. The averaged mass conservation equation for compressible flows can then be written

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial \bar{\rho} \tilde{v}_i}{\partial x_i} = 0. \quad (4.58)$$

Next, we derive the compressible time-averaged momentum equations. Initially, these are given by

$$\frac{\partial}{\partial t} (\bar{\rho} v_i) + \frac{\partial}{\partial x_j} (\bar{\rho} v_j v_i) = -\frac{\partial \bar{p}}{\partial x_i} + \mu \frac{\partial^2}{\partial x_j^2} (\bar{v}_i). \quad (4.59)$$

The unsteady term can be computed using Equation 4.47. We pay special attention to the convective term, which we expand according to Equation 4.52

$$\bar{\rho} v_j v_i = \bar{\rho} \tilde{v}_j \tilde{v}_i + \overline{\rho v_j'' v_i''}. \quad (4.60)$$

The time-averaged compressible momentum equations can be written

$$\frac{\partial}{\partial t} (\bar{\rho} \tilde{v}_i) + \frac{\partial}{\partial x_j} (\bar{\rho} \tilde{v}_j \tilde{v}_i + \overline{\rho v_j'' v_i''}) = -\frac{\partial \bar{p}}{\partial x_i} + \mu \frac{\partial^2}{\partial x_j^2} (\bar{v}_i). \quad (4.61)$$

where single-variable terms such as the pressure and diffusive components follow the same procedure described in Equation 4.57. Similar to the incompressible case, we rearrange some terms in the equation, such that

$$\frac{\partial}{\partial t} (\bar{\rho} \tilde{v}_i) + \frac{\partial}{\partial x_j} (\bar{\rho} \tilde{v}_j \tilde{v}_i) = -\frac{\partial \bar{p}}{\partial x_i} + \mu \frac{\partial}{\partial x_j} \left(\bar{\tau}_{ji} - \overline{\rho v_i'' v_j''} \right). \quad (4.62)$$

where $-\overline{\rho v_i'' v_j''}$ is the Reynolds stress, which includes the instantaneous density ρ , and τ_{ij} is the viscous stress tensor. As explained in the incompressible section, the resulting system of equations remains underdetermined due to the Reynolds stresses. Obtaining an equation for these stresses turns out to introduce additional unknowns. We now discuss and derive equations for the Reynolds-stress tensor for incompressible flow.

4.2.2 The Reynolds Stresses

The Reynolds stresses for the incompressible case are given by

$$\hat{\tau}_{ij} = -\overline{\rho v_i' v_j'}. \quad (4.63)$$

We will show in the next section that we can't solve for them directly. However, for now we can understand their influence on the mean flow. Recall the integral form of the Navier-Stokes equations from Chapter 2, which we rewrite

$$\frac{d}{dt} \int_{\Omega} \rho v_i d\Omega + \oint_S \rho v_i \vec{v} \cdot d\vec{s} = \oint_S -P d\vec{s} + \oint_S \tau_{ij} d\vec{s}. \quad (4.64)$$

The term on the LHS represents the rate of change of momentum. On the right-hand side of Equation 4.64, the first term is responsible for the convection of momentum in and out of the system, followed by the pressure term, and finally the viscous diffusion of momentum. When we apply our time-averaging to this equation, we obtain

$$\frac{d}{dt} \int_{\Omega} \rho \bar{v}_i d\Omega + \oint_S \rho \bar{v}_i \bar{\vec{v}} \cdot d\vec{s} = \oint_S -\bar{P} d\vec{s} + \oint_S (\bar{\tau}_{ij} - \overline{\rho v_i' v_j'}) d\vec{s}. \quad (4.65)$$

Equation 4.65 is similar to the time-averaged systems we have derived in the previous section in differential form, such as Equations 4.43 and 4.62. Note that the new Reynolds stress term naturally lumps together with the viscous stresses. In fact, the Reynolds stresses are the *turbulent diffusion* of momentum into Ω .

Example

Consider laminar and turbulent flow over a plate that is heated, such as that shown in Figure 4.6. By taking the average of the turbulent case, the resulting profile will resemble that in Figure 4.7. If we now compare the heat transfer between the time-averaged turbulent case and the laminar profile, we will observe much more heat transfer in the time-averaged profile. The reason is due to the energy transported by the turbulent eddies, which after time-averaging looks like extra diffusion.

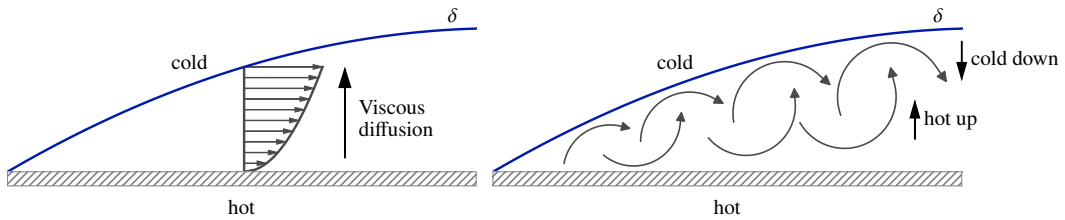


Figure 4.6: Reynolds stresses example

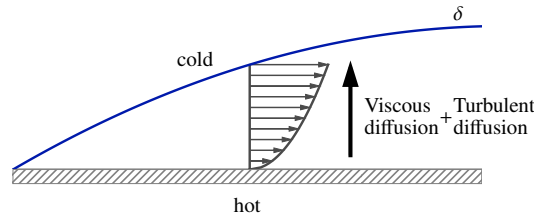


Figure 4.7: Reynolds stresses example

4.2.3 The RANS Closure Problem

As we have previously shown, the Reynolds stresses stem from averaging the convective terms of the Navier-Stokes equations. We now need to find additional equations to solve our system of time-averaged conservation laws. Specifically, we would like to define equations that describe the rate of change of these stresses as we have done for other flow quantities. In this section, we derive the expressions for incompressible flow, however, the derivation can be readily performed for the compressible case considering the Favre averaging procedure.

Consider the incompressible momentum equation in the i -th direction, which we conveniently rewrite

$$\rho \frac{\partial v_i}{\partial t} + \rho v_k \frac{\partial v_i}{\partial x_k} + \frac{\partial p}{\partial x_i} - \mu \frac{\partial^2 v_i}{\partial x_k \partial x_k} = 0. \quad (4.66)$$

Since we are looking for a time-averaged symmetric tensor for the Reynolds stresses, we perform the following operation

$$\overline{v'_i P(v_j)} + \overline{v'_j P(v_i)} = 0. \quad (4.67)$$

Here, v'_i and v'_j are fluctuation components in the i -th and j -th direction, respectively. We can compactly rewrite Equation 4.67 using

$$A_{ij} + B_{ij} + C_{ij} + D_{ij} = 0, \quad (4.68)$$

where A_{ij} , B_{ij} , C_{ij} , D_{ij} are the unsteady, convective, pressure and viscous stress tensors from the momentum equations, respectively. For the sake of clarity, we expand each of these terms

$$A_{ij} = \overline{v'_i \rho \frac{\partial v_j}{\partial t}} + \overline{v'_j \rho \frac{\partial v_i}{\partial t}}, \quad (4.69)$$

$$B_{ij} = \overline{v'_i \rho v_k \frac{\partial v_j}{\partial x_k}} + \overline{v'_j \rho v_k \frac{\partial v_i}{\partial x_k}}, \quad (4.70)$$

$$C_{ij} = \overline{v'_j \frac{\partial p}{\partial x_i}} + \overline{v'_i \frac{\partial p}{\partial x_j}}, \quad (4.71)$$

$$D_{ij} = -\overline{v'_i \mu \frac{\partial^2 v_j}{\partial x_k \partial x_k}} - \overline{v'_j \mu \frac{\partial^2 v_i}{\partial x_k \partial x_k}}. \quad (4.72)$$

We will now time-average and derive an expression for the Reynolds-stress tensor. Initially, we want to split the instantaneous variables using Reynolds decomposition. Starting with the unsteady term

$$A_{ij} = \overline{\rho v'_i \frac{\partial}{\partial t} (\bar{v}_j + v'_j)} + \overline{\rho v'_j \frac{\partial}{\partial t} (\bar{v}_i + v'_i)}, \quad (4.73)$$

which we rewrite after solving the products

$$A_{ij} = \overline{\rho v'_i \frac{\partial \bar{v}_j}{\partial t}} + \overline{\rho v'_i \frac{\partial v'_j}{\partial t}} + \overline{\rho v'_j \frac{\partial \bar{v}_i}{\partial t}} + \overline{\rho v'_j \frac{\partial v'_i}{\partial t}}. \quad (4.74)$$

Recall from the Reynolds averaging properties in Equation 4.28 that the average of the product of a mean and a fluctuating quantity is zero. Hence, we eliminate the first and third terms in Equation 4.74. Furthermore, we apply the product rule to bring fluctuation quantities into the temporal derivative. This yields the time-averaged convective term

$$A_{ij} = \overline{\rho v'_i \frac{\partial v'_j}{\partial t}} + \overline{\rho v'_j \frac{\partial v'_i}{\partial t}} = \rho \frac{\partial}{\partial t} (\overline{v'_i v'_j}). \quad (4.75)$$

Next, we decompose the variables in the convective term

$$B_{ij} = \overline{\rho v'_i (\bar{v}_k + v'_k) \frac{\partial}{\partial x_k} (\bar{v}_j + v'_j)} + \overline{\rho v'_j (\bar{v}_k + v'_k) \frac{\partial}{\partial x_k} (\bar{v}_i + v'_i)}, \quad (4.76)$$

whereby solving the products, we obtain

$$\begin{aligned} B_{ij} = & \overline{v'_i \bar{v}_k \frac{\partial \bar{v}_j}{\partial x_k}} + \overline{v'_i \bar{v}_k \frac{\partial v'_j}{\partial x_k}} + \overline{v'_i v'_k \frac{\partial \bar{v}_j}{\partial x_k}} + \overline{v'_i v'_k \frac{\partial v'_j}{\partial x_k}} \\ & + \overline{v'_j \bar{v}_k \frac{\partial \bar{v}_i}{\partial x_k}} + \overline{v'_j \bar{v}_k \frac{\partial v'_i}{\partial x_k}} + \overline{v'_j v'_k \frac{\partial \bar{v}_i}{\partial x_k}} + \overline{v'_j v'_k \frac{\partial v'_i}{\partial x_k}}. \end{aligned} \quad (4.77)$$

Note that $\overline{u' \bar{v} \bar{w}} = \overline{u' v w} = 0$ as a consequence of Equation 4.28. This way we can clean up Equation 4.77 by eliminating the first and fifth terms. In addition, using the product rule for the second and fourth term, we have

$$\overline{v'_i \bar{v}_k \frac{\partial v'_j}{\partial x_k}} + \overline{v'_j \bar{v}_k \frac{\partial v'_i}{\partial x_k}} = \bar{v}_k \frac{\partial}{\partial x_k} (\overline{v'_i v'_j}), \quad (4.78)$$

where \bar{v}_k is a constant. This yields

$$B_{ij} = \rho \left[\bar{v}_k \frac{\partial}{\partial x_k} (\overline{v'_i v'_j}) + \overline{v'_i v'_k \frac{\partial \bar{v}_j}{\partial x_k}} + \overline{v'_k \frac{\partial}{\partial x_k} (v'_i v'_j)} + \overline{v'_j v'_k \frac{\partial \bar{v}_i}{\partial x_k}} \right]. \quad (4.79)$$

We can use the chain rule on the third term of Equation 4.79, such that

$$\frac{\partial}{\partial x_k} (\overline{v'_i v'_j v'_k}) = \overline{v'_i v'_j \frac{\partial v'_k}{\partial x_k}} + \overline{v'_k \frac{\partial}{\partial x_k} (v'_i v'_j)} = \overline{v'_k \frac{\partial}{\partial x_k} (v'_i v'_j)}, \quad (4.80)$$

where the first term is zero due to continuity (see Equation 4.34). We can now write the time-averaged convective term of the Reynolds stress equation

$$B_{ij} = \bar{v}_k \frac{\partial}{\partial x_k} (\rho \overline{v'_i v'_j}) + \rho \overline{v'_i v'_k \frac{\partial \bar{v}_j}{\partial x_k}} + \frac{\partial}{\partial x_k} (\rho \overline{v'_i v'_j v'_k}) + \rho \overline{v'_j v'_k \frac{\partial \bar{v}_i}{\partial x_k}}. \quad (4.81)$$

Now, we move on to the pressure term. By splitting the instantaneous quantities, we obtain

$$C_{ij} = \overline{v'_i \frac{\partial}{\partial x_j} (\bar{p} + p')} + \overline{v'_j \frac{\partial}{\partial x_i} (\bar{p} + p')}, \quad (4.82)$$

which after solving the products and cancelling the linear terms yields

$$C_{ij} = \overline{v'_i \frac{\partial p'}{\partial x_j}} + \overline{v'_j \frac{\partial p'}{\partial x_i}}. \quad (4.83)$$

Finally, we decompose the viscous term

$$D_{ij} = -\mu \left[\overline{v'_i \frac{\partial^2}{\partial x_k \partial x_k} (\bar{v}_j + v'_j)} + \overline{v'_j \frac{\partial^2}{\partial x_k \partial x_k} (\bar{v}_i + v'_i)} \right], \quad (4.84)$$

whereby solving the products and cancelling corresponding terms yields

$$D_{ij} = -\mu \left[\overline{v'_i \frac{\partial^2 v'_j}{\partial x_k \partial x_k}} + \overline{v'_j \frac{\partial^2 v'_i}{\partial x_k \partial x_k}} \right]. \quad (4.85)$$

From the chain rule, we can write

$$\frac{\partial^2}{\partial x_k \partial x_k} (\overline{v'_i v'_j}) = 2 \frac{\partial \overline{v'_i}}{\partial x_k} \frac{\partial \overline{v'_j}}{\partial x_k} + \overline{v'_i \frac{\partial^2 v'_j}{\partial x_k \partial x_k}} + \overline{v'_j \frac{\partial^2 v'_i}{\partial x_k \partial x_k}}, \quad (4.86)$$

which yields the time-averaged viscous term

$$D_{ij} = -\mu \frac{\partial^2}{\partial x_k \partial x_k} (\overline{v'_i v'_j}) + 2\mu \frac{\partial \overline{v'_i}}{\partial x_k} \frac{\partial \overline{v'_j}}{\partial x_k}. \quad (4.87)$$

The resulting equation for the evolution of the Reynolds stresses can be written as

$$\begin{aligned} \frac{\partial}{\partial t} (\rho \overline{v'_i v'_j}) + \bar{v}_k \frac{\partial}{\partial x_k} (\rho \overline{v'_i v'_j}) = & -\rho \overline{v'_i v'_k} \frac{\partial \bar{v}_j}{\partial x_k} - \rho \overline{v'_j v'_k} \frac{\partial \bar{v}_i}{\partial x_k} - \overline{v'_i \frac{\partial p'}{\partial x_j}} - \overline{v'_j \frac{\partial p'}{\partial x_i}} \\ & + \frac{\partial^2}{\partial x_k^2} \left[\mu \frac{\partial}{\partial x_k} (\overline{v'_i v'_j}) - (\rho \overline{v'_i v'_j v'_k}) \right] - 2\mu \frac{\partial \overline{v'_i}}{\partial x_k} \frac{\partial \overline{v'_j}}{\partial x_k}. \end{aligned} \quad (4.88)$$

By inspecting Equation 4.88, it is clear that we have added six equations, but the number of unknowns also increased. The high-order correlation $\overline{v'_i v'_j v'_k}$ is responsible for ten additional unknowns, in addition to other twelve that result from the new pressure and viscosity terms. This is known as the *closure* problem of turbulence. Interestingly, if we decided to further find equations by taking additional moments, new higher-order correlations would appear. This is a consequence of the initial averaging of the Navier-Stokes equations. By multiplying Equation 4.88 by $-\rho^{-1}$ and using the following definitions

$$\hat{\tau}_{ij} = -\overline{v'_i v'_j}, \quad (4.89)$$

$$\Pi_{ij} = \frac{p'}{\rho} \left(\frac{\partial \overline{v'_i}}{\partial x_j} + \frac{\partial \overline{v'_j}}{\partial x_i} \right), \quad (4.90)$$

$$\varepsilon_{ij} = 2\nu \frac{\partial \overline{v'_i}}{\partial x_k} \frac{\partial \overline{v'_j}}{\partial x_k}, \quad (4.91)$$

$$\rho C_{ijk} = \rho \overline{v'_i v'_j v'_k} + \overline{p' v'_i} \delta_{jk} + \overline{p' v'_j} \delta_{ik}, \quad (4.92)$$

we can compactly write the Reynolds-stress equation in its common form

$$\frac{\partial \hat{\tau}_{ij}}{\partial t} + \bar{v}_k \frac{\partial \hat{\tau}_{ij}}{\partial x_k} = -\hat{\tau}_{ij} \frac{\partial \bar{v}_j}{\partial x_k} - \hat{\tau}_{jk} \frac{\partial \bar{v}_i}{\partial x_k} + \varepsilon_{ij} - \Pi_{ij} + \frac{\partial}{\partial x_k} \left(\nu \frac{\partial \hat{\tau}_{ij}}{\partial x_k} + C_{ijk} \right). \quad (4.93)$$

We are still left with a system of equations that remains to be closed. Hence, we have shown that the Reynolds stresses cannot be found without solving the full unsteady Navier-Stokes equations. In the quest of solving turbulent flows using the RANS equations, we dedicate the rest of this chapter to the derivation of common empirical models that have been devised to estimate $\hat{\tau}_{ij}$.

4.3 Turbulence Modelling

In the previous section, we derived the RANS equations. This time-averaged system of conservation laws is widely used in the aerospace industry, in particular, due to its relatively low computational cost when compared to unsteady simulations of the Navier-Stokes equations. We have also stated that their use is limited to obtaining mean flow properties due to the averaging of the turbulent scales. In addition, we have shown that we require additional mathematical equations to give closure to the system. This is done by introducing approximations that model features of the Reynolds stress tensor. We will observe that none of these models is applicable to all types of flow. Indeed, each of these turbulence models has its own typical applications and known limitations. We begin by presenting the Boussinesq approximation, where a new apparent viscous parameter known as the *eddy viscosity* is introduced. Later, it will be shown that the majority of popular RANS models devise different strategies to model this parameter, which is accomplished in part by tuning parameters after comparison with experimental data.

We categorize these models according to the number of additional partial differential equations required for turbulence closure. We present zero-equation, also known as algebraic, one-equation and two-equation models. Note that models with a larger number of PDEs exist; however, this also implies a decrease in computational efficiency. A second feature that categorizes these models, according to Wilcox [25] is whether the model can be considered *complete* or *incomplete*. A complete model does not require previously known information about the turbulence properties of the flow to simulate, whereas incomplete models do. We also note that generally, one must be careful when choosing a turbulence model. While some may be used to predict turbulence mean properties, others require previous calibration and are rarely used for unknown flows. We note that this section includes some of the most known turbulence models, and it is by no means a turbulence modelling reference by itself.

4.3.1 The Boussinesq Hypothesis

In 1877, Boussinesq presented an approximation for the turbulent transfer of momentum in relation to molecular motion. Boussinesq observed that the effect of turbulence, when time-averaged, is similar to the effects of viscosity. Turbulent fluctuations transport conserved quantities, such as mass, momentum, and energy, by physically moving it around. When this is time-averaged, the effect of turbulence is to spread things out, similar to the diffusive terms in the Navier-Stokes equations. Hence, Boussinesq proposed, based on the transfer of momentum in kinetic theory of gases, replacing the Reynolds stress tensor $\hat{\tau}_{ij}$ by simply adding extra turbulent viscosity. His hypothesis can be written for three-dimensional flows as

$$-\rho \overline{v'_i v'_j} = \rho \hat{\tau}_{ij} = 2\mu_\tau \left(\bar{S}_{ij} - \frac{1}{3} \frac{\partial \bar{v}_k}{\partial x_k} \delta_{ij} \right) - \frac{2}{3} k \delta_{ij}, \quad (4.94)$$

where

$$\bar{S}_{ij} = \frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i}, \quad (4.95)$$

k is the kinetic energy $\frac{1}{2} \overline{v'_i v'_i}$, δ_{ij} is the Kronecker delta and μ_τ is an isotropic scalar quantity, known as the *eddy viscosity*. This value is a function of the local flow, and not of the fluid itself. Note

that for incompressible flows, $\frac{\partial \bar{v}_k}{\partial x_k} \equiv 0$ due to continuity. This closing approximation of the RANS equations is the basis of the turbulence models presented hereafter. Now, the remaining task is to define the value of μ_τ , which has reduced all of the unknown Reynolds stress terms down to a single scalar unknown.

Observe that the resulting Reynolds stresses are now represented as additional viscous terms. This is, in fact, contradictory since we are neglecting the convective effects of these stresses. Is this a real contradiction? Indeed, the assumption that the Reynolds stress tensor behaves linearly with the rate of strain is incorrect for most flows and is the result of incorrectly associating the behaviour of gas molecules from the kinetic theory of gases to that of the turbulent structures. In fact, turbulent transfer of momentum cannot be associated with the molecular transfer of momentum in gases. One reason is that eddies cannot be assumed to be rigid bodies. Furthermore, recall that the size of turbulent structures ranges from the characteristic length of the domain to a tiny number that cubically decreases with the Reynolds number. Hence the relation between eddy sizes and their "mean free-path" cannot be associated with that of the molecules. These discrepancies show a lack of theoretical justification for the Boussinesq assumption. However, most turbulence models have been devised under this premise, and yield relatively good results. Beware, however, that models based on this assumption generally fail for highly anisotropic flows, such as boundary layers.

We will present a limited selection of popular turbulence models that leverage the eddy viscosity approximation presented by Boussinesq, commonly known as *eddy-viscosity models*.

4.3.2 The Mixing Length Model

Rather than defining a constant value of viscosity dependent on the velocity field, Prandtl proposed a *rough* approach for the calculation of the kinematic eddy viscosity $\nu_\tau = \mu_\tau / \rho$ [16]. Recall the mean-free-path in a gas is the distance travelled by a molecule before it collides with another. When this collision happens, momentum transfer occurs. Analogous to this distance, Prandtl considered a *mixing length* ℓ_m , which is equivalent to the path travelled by a structure in the flow, such as a vortex, before losing momentum due to mixing with a neighbouring structure. This length was typically taken to be constant and corresponds to the size of the dominant turbulent eddies in the flow. In one dimension, we note that ν_τ has SI units m^2/s . Hence, by performing a dimensional analysis, it seems a natural fit to use

$$\nu_\tau = \ell_m v^*, \quad (4.96)$$

where v^* is the corresponding characteristic velocity. We can determine the value of v^* , which Prandtl proposed to be

$$v^* \propto \ell_m \left| \frac{\partial \bar{v}_x}{\partial y} \right|. \quad (4.97)$$

Hence, the characteristic velocity and ℓ_m could be related to the eddy viscosity by

$$\nu_\tau = \ell_m^2 \left| \frac{\partial \bar{v}_x}{\partial y} \right|, \quad (4.98)$$

where ℓ_m is empirically determined and is known for only certain types of flows. Prandtl postulated that near solid walls, the velocity of the flow is proportional to the distance from the surface. This is consistent with the observed behaviour of flows within $\approx 10\%$ of their height. A profile of a typical boundary layer is shown in Figure 4.8.

We have now algebraically defined Boussinesq's viscosity μ_τ with another empirical parameter ℓ_m . The specification of this mixing length typically relies on previous experimental data and is dependent on whether it is a jet, wake, wall boundary layer or another type of flow [1]. Therefore, this model is considered to be incomplete, since it requires a previous understanding of the turbulent structures that are expected to be in the flow.

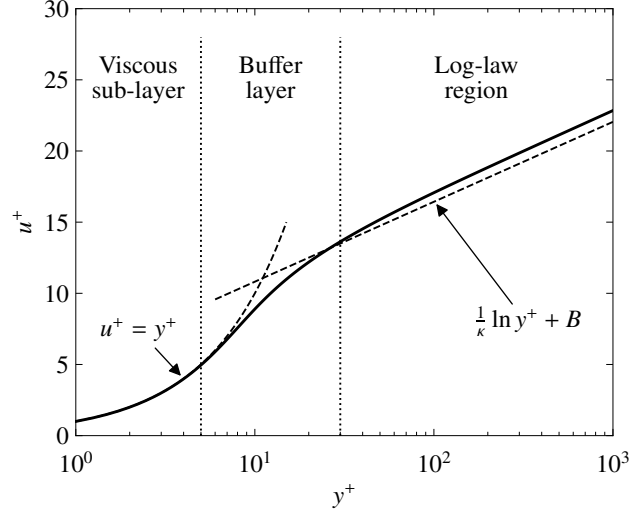


Figure 4.8: Behaviour of the mean velocity profile in wall-bounded turbulent flows

4.3.3 The Spalart-Allmaras Model

Spalart and Allmaras [20] (SA) took an empirical approach, using arguments from dimensional analysis to develop a new model, which includes the resolution of an ad-hoc transport equation for a turbulence variable \tilde{v} . The model was initially developed for aerodynamics applications such as transonic flow over airfoils. Several nested versions are included in the original publication, which range from simple models for free shear flows to more complex derivations that can be applied to boundary layer flows. The general model is given by

$$\frac{\partial \tilde{v}}{\partial t} + v_i \frac{\partial \tilde{v}}{\partial x_i} = c_{b1}(1 - f_{t2})\tilde{S}\tilde{v} + \frac{1}{\sigma} \left[\frac{\partial}{\partial x_j} \cdot \left((v + \tilde{v}) \frac{\partial \tilde{v}}{\partial x_j} \right) + c_{b2} \left(\frac{\partial \tilde{v}}{\partial x_j} \right)^2 \right] - \left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2} \right], \quad (4.99)$$

where c and f are empirical constants and functions of the turbulence model, which will be defined shortly along with the rest of the variables. Upon solving Equation 4.99, we can compute the eddy viscosity μ_τ from

$$\mu_\tau = \rho \tilde{v} f_{v1}, \quad \chi = \frac{\tilde{v}}{\nu}, \quad (4.100)$$

where ν is the molecular kinematic viscosity. In addition,

$$\tilde{S} \equiv S + \frac{\tilde{v}}{\kappa^2 d^2} f_{v2}, \quad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \quad f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}}, \quad (4.101)$$

with S the magnitude of the vorticity and d the distance to the closest wall [20].

$$f_w = g \left[\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6} \right]^{1/6}, \quad r \equiv \min \left(\frac{\tilde{v}}{\tilde{S} \kappa^2 d^2}, 10 \right). \quad (4.102)$$

$$f_{t2} = c_{t3} \exp[-c_{t4} \chi^2] \quad g = [r + c_{w2}(r^6 - r)]. \quad (4.103)$$

Finally, the constants are given by

$$c_{b1} = 0.1355, \quad c_{b2} = 0.622, \quad \sigma = 2/3, \quad \kappa = 0.41, \quad (4.104)$$

$$c_{w2} = 0.3, \quad c_{w3} = 2, \quad c_{v1} = 7.1, \quad c_{t3} = 1.2, \quad (4.105)$$

$$c_{t4} = 0.5, \quad c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma}. \quad (4.106)$$

The implementation of this model allows us to remove the incomplete condition from the previously defined models. Wall and freestream boundary conditions are [18]

$$\tilde{v}_{wall} = 0, \quad \tilde{v}_{\infty} = 3v_{\infty} : to : 5v_{v_{\infty}}, \quad (4.107)$$

Now, by solving Equations 4.99 and 4.100, we are able to give closure to the RANS equations. Since only one additional equation is required, the relative computational cost of this turbulence model makes it appealing. Given that this model was initially developed for aerospace applications, it is a popular choice for the prediction of aerodynamic loads at low to moderate angles of attack due to its relatively good resolution of boundary layer problems.

Furthermore, the SA model is well-suited for applications using unstructured meshes. Despite its success in predicting aircraft performance, it has been found to unsuitable for flows that include jet-like free-shear regions and flows involving complex recirculation.

4.3.4 The k-ε Model

One of the most commonly used RANS models for the computation of turbulent flows is the $k - \varepsilon$ model. It was originally developed to overcome the deficiencies of the mixing length model, such that no specification of flow-dependent properties are required. Instead, closure is achieved by resolving two transport equations for the turbulent kinetic energy k and its rate of dissipation ε . Hence this model is complete. Different versions of this model have been provided [9, 12]. We describe the *standard* $k - \varepsilon$ model, as shown by [2]. The relation between the eddy-viscosity and the two aforementioned turbulent quantities is given by

$$\mu_{\tau} = c_{\mu} \rho k^2 / \varepsilon, \quad (4.108)$$

where c_{μ} is a constant, which we define at the end of this section. Define the instantaneous kinetic energy to be the sum of the mean and the fluctuation component, such that

$$\begin{aligned} k(t) &= \bar{k} + k', \\ &= \frac{1}{2} \sum_{i=1}^d \bar{v}_i + \frac{1}{2} \sum_{i=1}^d \overline{v'_i v'_i}. \end{aligned} \quad (4.109)$$

Hence, by multiplying each velocity component with their corresponding momentum equation in the appropriate direction, a PDE for the turbulent kinetic energy can be obtained. After performing Reynolds averaging and several algebraic operations, we can write

$$\rho \frac{\partial k}{\partial t} + \rho \bar{v}_j \frac{\partial k}{\partial x_j} = \rho \hat{\tau}_{ij} \frac{\partial \bar{v}_i}{\partial x_j} - \rho \varepsilon + \frac{\partial}{\partial x_j} \left[(\mu + \mu_{\tau} / \sigma_k) \frac{\partial k}{\partial x_j} \right]. \quad (4.110)$$

The second component of this model is the dissipation rate ε , for which the following transport equation was derived

$$\rho \frac{\partial \varepsilon}{\partial t} + \rho \bar{v}_j \frac{\partial \varepsilon}{\partial x_j} = c_{\varepsilon 1} \frac{\varepsilon}{k} \rho \hat{\tau}_{ij} \frac{\partial \bar{v}_i}{\partial x_j} - \rho c_{\varepsilon 2} \frac{\varepsilon^2}{k} + \frac{\partial}{\partial x_j} \left[(\mu + \mu_{\tau} / \sigma_{\varepsilon}) \frac{\partial \varepsilon}{\partial x_j} \right], \quad (4.111)$$

where the terms on the right-hand side refer to the diffusion, production and dissipation rates of ε [1]. The constants are given by

$$c_\mu = 0.09, \quad c_{\varepsilon 1} = 1.44, \quad c_{\varepsilon 2} = 1.92, \quad (4.112)$$

$$\sigma_k = 1.0, \quad \sigma_\varepsilon = 1.3. \quad (4.113)$$

The computational cost that comes from the computation of two transport equations can be significant. However, as previously mentioned, this model is one of the most commonly used in the CFD field, mostly for the modelling of free-shear layer flows with small to moderate pressure gradients. One of the pitfalls of this model, however, lies in the prediction of flows with large adverse pressure gradients, which cause the model to predict results inaccurately. This means that this model is often inaccurate for turbulent wall-bounded flows. Additional modifications, such as the introduction of wall functions, are necessary for such applications. Nevertheless, this model remains one of the simplest and most popular complete models.

4.3.5 The $k-\omega$ Model

The $k-\omega$ model is also a two-equation model, and along with the $k-\varepsilon$ model, it is one of the most commonly used turbulence RANS models. Here, the equation for the turbulent kinetic energy is the same as in the $k-\varepsilon$. The second transport equation, however, is derived for the specific dissipation rate, $\omega \equiv \varepsilon/k$. The eddy viscosity is defined to be

$$\mu_\tau = \rho \frac{k}{\tilde{\omega}}, \quad \tilde{\omega} = \max \left[\omega, \frac{7}{8} \sqrt{2S_{ij}S_{ij}/\beta^*} \right]. \quad (4.114)$$

For the sake of completeness, we rewrite the PDE for the kinetic energy, now in terms of ω

$$\rho \frac{\partial k}{\partial t} + \rho \bar{v}_j \frac{\partial k}{\partial x_j} = \rho \hat{\tau}_{ij} \frac{\partial \bar{v}_i}{\partial x_j} - \rho \beta^* k \omega + \frac{\partial}{\partial x_j} \left[\left(\mu + \rho \sigma^* \frac{k}{\omega} \right) \frac{\partial k}{\partial x_j} \right]. \quad (4.115)$$

The specific dissipation rate ω can be computed through [25]

$$\rho \frac{\partial \omega}{\partial t} + \rho \bar{v}_j \frac{\partial \omega}{\partial x_j} = \alpha \frac{\omega}{\kappa} \rho \hat{\tau}_{ij} \frac{\partial \bar{v}_i}{\partial x_j} - \rho \beta \omega^2 + \rho \frac{\sigma_d}{\omega} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j} + \frac{\partial}{\partial x_j} \left[\left(\mu + \rho \sigma^* \frac{k}{\omega} \right) \frac{\partial \omega}{\partial x_j} \right]. \quad (4.116)$$

The closure coefficients and auxiliary relations are given by

$$\alpha = \frac{13}{25}, \quad \beta = \beta_o f_\beta, \quad \beta^* = \frac{9}{100}, \quad (4.117)$$

$$\sigma = \frac{1}{2}, \quad \sigma^* = \frac{3}{5}, \quad \sigma_{do} = \frac{1}{8}, \quad (4.118)$$

$$\sigma_d = \begin{cases} 0, & \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j} \leq 0, \\ \sigma_{do}, & \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j} > 0. \end{cases}$$

$$\beta_o = 0.0708, \quad f_\beta = \frac{1 + 85\chi_\omega}{1 + 100\chi_\omega}, \quad \chi_\omega \equiv \left| \frac{\Omega_{ij}\Omega_{jk}S_{ki}}{(\beta^*\omega)^3} \right|, \quad (4.119)$$

$$\varepsilon = \beta^* \omega k, \quad (4.120)$$

where Ω_{ij} is the mean-rotation tensor

$$\Omega_{ij} = \frac{1}{2} \left(\frac{\partial \bar{v}_i}{\partial x_j} - \frac{\partial \bar{v}_j}{\partial x_i} \right), \quad (4.121)$$

and S_{ij} is the mean strain rate tensor, as described in Equation 4.95.

An advantage of this model is the relatively accurate resolution of near-wall interaction, and for flows with the presence of strong vorticity. This makes the $k - \omega$ approach suitable for applications such as turbomachinery. However, it was found to overpredict flow separation and to be highly sensitive to the freestream conditions. In addition, compared to the $k - \varepsilon$ model, it is not an appropriate choice for free-shear flows. Further improvements to this model have been made to accommodate its shortcomings [25]. Taking advantage of the relative accuracy of both the $k - \varepsilon$ and $k - \omega$ models, combinations of these have been derived, such as the Shear Stress Transport (SST) model. We refer the reader to NASA's Turbulence Resource website for further details and additional turbulence models [18].

4.3.6 Summary of Turbulence Models

We have introduced the basis of turbulence modelling using the RANS equations. As previously stated, these models are initially based on the Boussinesq approximation, where a constant coefficient referred to as the eddy viscosity, is used to model the behaviour of turbulence in a time-averaged sense. Further improvements, from the definition of a mixing length distance to the resolution of transport equations for turbulent quantities, have allowed the development of turbulence models to become applicable for real-world industrial applications. These models have been devised for specific applications and can also result in inaccurate results if not chosen appropriately. It is important to remember that these models are primarily limited by two approximations. First, the time-averaging of the Navier-Stokes equations removed the wide range of actual length scales characteristic of turbulent flows, which introduced the Reynolds stresses. Then, under the Boussinesq approximation, it was assumed that these Reynolds stresses could be reduced to a single unknown eddy viscosity, which is not always accurate. Then, this eddy viscosity is itself modelled using a turbulence model, which approximates the eddy viscosity to varying levels of realism.

To summarize the previous models, we present a table where we display the characteristic variables that define the model, the complete or incomplete condition according to [25], the number of equations to solve, and a general description including some common applications of each model. More complex models can be found in the literature. These include not only additional eddy-viscosity models but also other types of models such as Large Eddy Simulation (LES), which do not require a time-averaged RANS system of equations. For further references, refer to the NASA Turbulence Resource database [18].

Table 4.1: Summary of popular turbulence models.

Model	Variable(s)	Classification	Description
ML	A mixing length ℓ_m	Incomplete, algebraic model	An empirical mixing length is used to compute the eddy viscosity. This model was developed in a "rough" sense. Value of ℓ_m depends on the type of flow.
SA	Turbulent quantity proportional to the viscosity $\tilde{\nu}$	Complete, one-equation model	Resolution of a transport equation for a turbulent quantity, specifically a viscosity-like parameter. Computationally efficient since only a single transport equation is added. Applicable to determine aerodynamic loads at moderate angles of attack.
$k - \varepsilon$	Kinetic energy and rate of dissipation, k, ε	Complete, two-equation model	Two transport equations are added, one for the turbulent kinetic energy and the other for its rate of dissipation. Applicable to free-shear layer flows. Gives inaccurate results for flows with large adverse pressure gradients.
$k - \omega$	Kinetic energy and specific dissipation k, ω	Complete, two-equation model	Requires the computation of the kinetic energy and specific dissipation rate transport equations. The model works well across the boundary layer of wall turbulent flows, hence widely used in the aerospace field. Inaccurate for free-shear flows.

5. Boundary Conditions

5.1 Wall Boundaries

5.1.1 Wall-Bounded Turbulence

In order to properly enforce the wall boundary conditions, it is necessary to have a computational mesh that is sufficiently fine in this region. To better understand the physics of the flow in this region, we will reconsider the incompressible RANS momentum equations of the form

$$\rho \frac{\partial \bar{v}_i}{\partial t} + \rho \frac{\partial}{\partial x_j} (\bar{v}_j \bar{v}_i) = -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j} (\bar{\tau}_{ji} - \rho \overline{v'_j v'_i}). \quad (5.1)$$

Assuming that a full time average has been computed, the temporal derivative can be omitted

$$\rho \frac{\partial}{\partial x_j} (\bar{v}_j \bar{v}_i) = -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j} (\bar{\tau}_{ji} - \rho \overline{v'_j v'_i}). \quad (5.2)$$

Furthermore, if we assume that the pressure gradient term is negligible and that the streamwise velocity varies primarily in the vertical direction, then the vertical momentum equation yields

$$v \frac{\partial \bar{u}}{\partial y} - \overline{u'v'} = v \left. \frac{\partial \bar{u}}{\partial y} \right|_{y=0}, \quad (5.3)$$

after integration. This implies that the rate of streamwise momentum transport away from the wall is constant through the boundary layer, and that it is equal to the viscous stress that the wall applies to the fluid. Furthermore, there are two mechanisms that are responsible for transporting this momentum in the boundary layer. The first term, $v \partial \bar{u} / \partial y$, is simply the steady viscous stress that applies due to the shear in the fluid moving up the boundary layer. In contrast, $-\overline{u'v'}$, are the Reynolds stresses. These are responsible for transporting momentum through the boundary layer by physically moving it. Upward velocity fluctuations tend to move slow-moving fluid near the wall up in the boundary layer. In contrast, downward velocity fluctuations tend to move fast-moving fluid near the free stream down in the boundary layer, transporting momentum. Based on physical observations, the boundary layer is typically divided into three distinct regions.

The Inner Layer

The inner layer is the region very close to the wall boundary. The no-slip boundary condition that forces the velocity towards zero and, as previously discussed, the fluctuating velocity components also vanish here. Hence, in the inner layer, Equation 5.5 reduces to

$$\nu \frac{\partial \bar{u}}{\partial y} = \nu \left. \frac{\partial \bar{u}}{\partial y} \right|_{y=0}. \quad (5.4)$$

In this region of the boundary layer, momentum is transported almost entirely by viscous effects.

The Outer Layer

In the outer regions of the boundary layer, the mean velocity gradient becomes negligible. Hence, the viscous effects can be omitted and Equation 5.5 reduces to

$$-\overline{u'v'} = \nu \left. \frac{\partial \bar{u}}{\partial y} \right|_{y=0}, \quad (5.5)$$

Hence, in this region of the boundary layer, momentum is transported almost entirely by turbulent fluctuations. Classical experiments have shown this is well-approximated via a log-law, as shown in Figure 4.8.

The Buffer Layer

The buffer layer is simply the region between the inner and outer layers. Neither the viscous nor the Reynolds stress terms dominate here, and hence Equation 5.5 remains unchanged.

In order to better understand the behaviour of a boundary layer, we will first non-dimensionalize it using parameters from within the boundary layer, resulting in the dimensionless *wall-units*. Starting with the shear stress at the wall and the density, we obtain the friction velocity

$$u^\tau = \sqrt{\frac{\tau_w}{\rho}}. \quad (5.6)$$

From this, we can now define a dimensionless height

$$y^+ = \frac{yu^\tau}{\nu}, \quad (5.7)$$

typically called “y plus”, and a dimensionless velocity

$$u^+ = u/u^\tau, \quad (5.8)$$

typically called “u plus”. A plot of a standard boundary layer profile, including these three regions, is given in Figure 4.8. This shows that, interestingly, the velocity profile far from the wall boundary can be well-approximated using a logarithmic profile.



Part 2: Numerics

6	Taylor-Series	53
7	Finite Difference Methods	56
7.1	The First Derivative	
7.2	A General Approach	
7.3	The Second Derivative	
7.4	Example Applications	
8	Finite Volume Methods	65
8.1	Derivation	
8.2	The Riemann Problem	
8.3	Example Applications	
8.4	Linear Hyperbolic Problems	
8.5	Nonlinear Hyperbolic Problems	
8.6	MUSCL Schemes	
9	Consistency, Stability, Convergence ..	95
9.1	Consistency	
9.2	Stability	
9.3	Convergence	
10	Spectral Properties	105
10.1	Dissipation Error	
10.2	Dispersion Error	
11	Modified Equation Analysis	110
11.1	Linear Advection	
11.2	General Observations	
12	Time-Stepping	113
12.1	Explicit	
12.2	Implicit	
13	Iterative Methods	121
13.1	Gaussian Elimination	
13.2	Jacobi Iteration	
13.3	Gauss Seidel Iteration	
13.4	Successive Over-Relaxation	
13.5	Assessing Convergence	
13.6	Multigrid	
14	Applications	130
14.1	An Euler Solver	
14.2	A Navier-Stokes Solver	

6. Taylor-Series

In the previous sections, we introduced the governing conservation laws that need to be solved to accurately predict the behaviour of fluids. These included the Navier-Stokes equations and the time-averaged RANS equations, with their associated turbulence models. However, to start, we will consider the simplified conservation laws we derived earlier, which were the linear advection, Burgers, and linear diffusion equations, and move on from there to the full Navier-Stokes equations. Starting with the simplified systems of equations

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad (6.1)$$

$$\frac{\partial u}{\partial t} + \frac{1}{2} \frac{\partial u^2}{\partial x} = 0, \quad (6.2)$$

$$\frac{\partial u}{\partial t} - \beta \frac{\partial^2 u}{\partial x^2} = 0. \quad (6.3)$$

We notice that the form of all of these equations is very similar. All three involve a time derivative combined with a coefficient multiplied by a spatial derivative. The Finite Difference Method (FDM) uses well-known concepts from applied mathematics, specifically Taylor-Series. Hence, before deriving the Finite Difference Method, we will start by reviewing some concepts from Taylor-Series.

Arising from Taylors' theorem, Taylor Series allows the value of a sufficiently smooth function at some point $x + \Delta x$ to be predicted by using the value of the solution at the point x along with knowledge of all derivatives at the point x .

Theorem 6.0.1 — Taylor's Theorem. Let $k \geq 1$ and letting $f(x)$ be smooth and differentiable k times then

$$f(x + \Delta x) = f(x) + \frac{\partial f}{\partial x} \frac{\Delta x^1}{1!} + \frac{\partial^2 f}{\partial x^2} \frac{\Delta x^2}{2!} + \dots + \frac{\partial^k f}{\partial x^k} \frac{\Delta x^k}{k!}, \quad (6.4)$$

which can be written compactly for an expansion truncated to k terms as

$$f(x + \Delta x) = \sum_{i=0}^k \frac{\partial^i f}{\partial x^i} \frac{\Delta x^i}{i!}. \quad (6.5)$$

While there are some limitations, particularly around the smoothness of the function and its derivatives, it is worth taking a moment to consider just how powerful Taylor series can be. It allows us to represent the entirety of a smooth function using the value of the solution and its derivatives at only one point in the domain. In addition, similar to Fourier series, we often find that a truncated expansion, which omits higher-order terms in the summation, is sufficient for many applications.

We will demonstrate the utility of Taylor series via example. Consider a simple sine wave

$$f(x) = \sin(x), \quad (6.6)$$

which is periodic on the interval $[-\pi, \pi]$. We first note that sine functions are sufficiently smooth for Taylor series to be applied up to an infinite number of derivatives. Now, we will explore the behaviour of the Taylor series as we add more terms. When we use a finite number of terms in the expansion, we are *truncating* all of the higher-order terms. We will form our expansion about the point $x = 0$. If we include just the first term and truncate all others, we obtain

$$f_0(\Delta x) = 0, \quad (6.7)$$

which is not a particularly accurate approximation of a sine wave except very close to the origin. However, as we start to add more terms we obtain the following expressions,

$$f_1(\Delta x) = \Delta x, \quad (6.8)$$

$$f_3(\Delta x) = \Delta x - \frac{\Delta x^3}{6}, \quad (6.9)$$

$$f_5(\Delta x) = \Delta x - \frac{\Delta x^3}{6} + \frac{\Delta x^5}{120}, \quad (6.10)$$

$$f_7(\Delta x) = \Delta x - \frac{\Delta x^3}{6} + \frac{\Delta x^5}{120} - \frac{\Delta x^7}{5040}, \quad (6.11)$$

$$f_9(\Delta x) = \Delta x - \frac{\Delta x^3}{6} + \frac{\Delta x^5}{120} - \frac{\Delta x^7}{5040} + \frac{\Delta x^9}{362880}, \quad (6.12)$$

where the subscript denotes the power of the highest degree expansion term included in the approximation. Each of these is a polynomial approximation of a sine wave about the point $x = 0$, and are plotted in Figure 6.1. We can make a few observations about the behaviour of Taylor series in general. First, when we are close to the expansion point $x = 0$, even approximations with a small number of terms are close to the true function. For example, the f_1 expansion is the well-known small-angle approximation of a sine wave. Then, as we add more terms, the accuracy of the expansion improves rapidly. For example, by ten terms, the approximation is nearly indistinguishable from the true function on the entire interval. Hence, we note that Taylor series becomes more accurate as we get closer to the expansion point and/or as we increase the number of terms.

To understand the behaviour of these truncated expansions, we note that their error arises from truncating the higher-order terms of the expansion. Furthermore, when Δx is relatively small, the error is dominated by the first truncated term since the higher-order terms rapidly approximate zero as Δx decreases. Hence, if we consider an infinite expansion

$$f(\Delta x) = \Delta x - \frac{\Delta x^3}{6} + \frac{\Delta x^5}{120} - \frac{\Delta x^7}{5040} + \frac{\Delta x^9}{362880} - \frac{\Delta x^{11}}{39916800} + \dots, \quad (6.13)$$

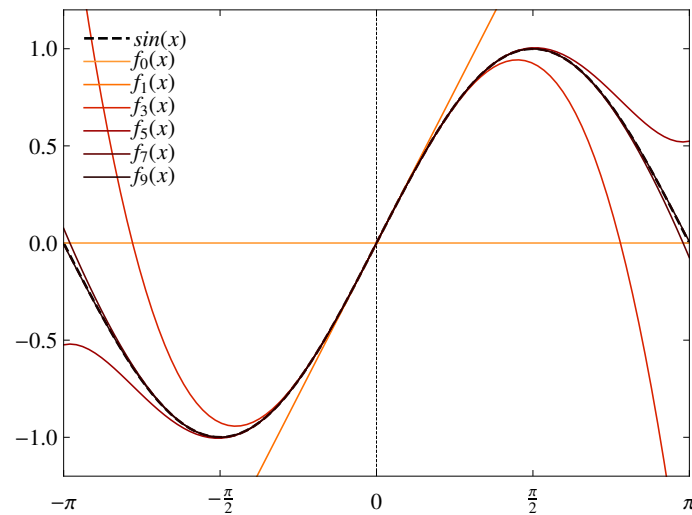


Figure 6.1: Taylor series expansion of $\sin(x)$ about $x = 0$

we note that the leading term omitted from the $f_1(\Delta x)$ approximation behaves like Δx^3 , the leading error term of the $f_1(\Delta x)$ approximation behaves like Δx^5 , and so on. Hence, as we get closer to the expansion point $x = 0$, we expect that the error shrinks, and that the rate at which the error shrinks will be proportional to the exponent of the leading truncated term of the expansion.



Check out the Burgers Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

7. Finite Difference Methods

7.1 The First Derivative

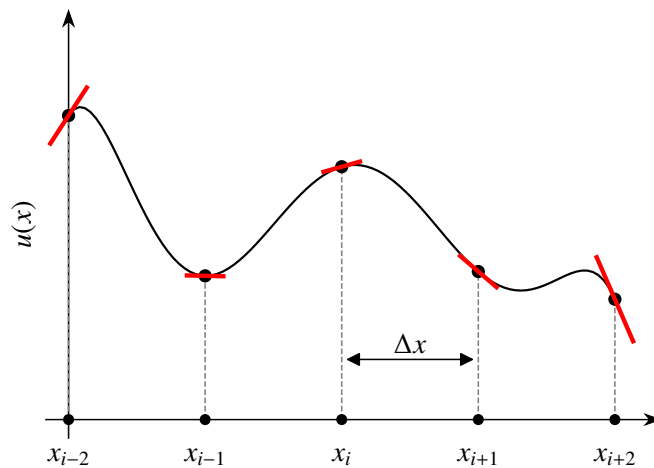


Figure 7.1: One-dimensional discretization using Finite Difference methods

To introduce the finite difference method, we start with a general function $u(x)$. In general, this function can be in any number of dimensions, but for simplicity we will first consider the one-dimensional case. We note that a function can generally be *approximated* by its values at a *discrete* set of points. An example of this is shown in Figure 7.1 where Δx is the grid spacing between the points. We now imagine that we are at some point i whose solution is $u_i = u(x_i)$ where

x_i is the physical location of the point. Using Taylor series, we can approximate the value of the solution one grid point to the left from the expansion

$$u_{i-1} = u_i - \Delta x \left. \frac{\partial u}{\partial x} \right|_{x_i} + \frac{\Delta x^2}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} - \frac{\Delta x^3}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^4), \quad (7.1)$$

where $\mathcal{O}(\Delta x^4)$ denotes the order of the next highest term in the expansion. We can also use another Taylor series to predict the value of the solution one point to the right

$$u_{i+1} = u_i + \Delta x \left. \frac{\partial u}{\partial x} \right|_{x_i} + \frac{\Delta x^2}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} + \frac{\Delta x^3}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^4). \quad (7.2)$$

So far, we have just applied Taylor series to try to determine the solution at points around our initial point u_i . However, if we look at Equation 7.1 we can use it in a different way. Rearranging we get

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = \frac{u_i - u_{i-1}}{\Delta x} + \frac{\Delta x}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} - \frac{\Delta x^2}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^3). \quad (7.3)$$

We note that as the grid spacing gets smaller, the terms with leading Δx , Δx^2 , and so on will tend to zero and the approximation

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = \frac{u_i - u_{i-1}}{\Delta x} + \mathcal{O}(\Delta x), \quad (7.4)$$

will converge to the true value of the derivative at the point x_i as Δx tends to zero. We can also do the same trick with Equation 7.2. Rearranging, we get

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = \frac{u_{i+1} - u_i}{\Delta x} - \frac{\Delta x}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} + \frac{\Delta x^2}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^3), \quad (7.5)$$

and again if the grid spacing is small, then the higher-order terms tend to zero and we are left with

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = \frac{u_{i+1} - u_i}{\Delta x} + \mathcal{O}(\Delta x), \quad (7.6)$$

which also allows us to approximate the derivative at x_i . Hence, if we know that value of the solution at x_i and either x_{i-1} or x_{i+1} we can approximate the derivative at x_i . We note that for both of these approximations, the error in the derivative approximation will decrease proportional to the grid spacing since the leading term in the truncation error is of $\mathcal{O}(\Delta x)$. We refer to a scheme of this form as a *first-order* finite difference method.

While the above examples are often useful, they converge rather slowly to the exact value of the derivative at the point x_i . However, if we take an average of Equation 7.3 and Equation 7.5, we obtain

$$\begin{aligned} \left. \frac{\partial u}{\partial x} \right|_{x_i} &= \frac{1}{2} \left(\frac{u_i - u_{i-1}}{\Delta x} + \frac{\Delta x}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} - \frac{\Delta x^2}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} \right) + \\ &\quad \frac{1}{2} \left(\frac{u_{i+1} - u_i}{\Delta x} - \frac{\Delta x}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} + \frac{\Delta x^2}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} \right) + \mathcal{O}(\Delta x^3) \end{aligned} \quad (7.7)$$

which simplifies to

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = \frac{u_{i+1} - u_{i-1}}{2\Delta x} - \frac{\Delta x^2}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^3), \quad (7.8)$$

noting that the leading truncation error terms conveniently cancelled each other out. Hence, if we assume that Δx is relatively small, we obtain

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = \frac{u_{i+1} - u_{i-1}}{2\Delta x} + \mathcal{O}(\Delta x^2). \quad (7.9)$$

Using this equation, we can approximate the value of the derivative of the function at x_i using the values of the solution at two neighbouring points x_{i-1} and x_{i+1} . However, unlike the previous finite difference methods, the truncation error of this approximation is $\mathcal{O}(\Delta x^2)$, meaning that as the grid spacing is reduced, the error will converge like the grid spacing squared. We refer to a scheme of this form as a *second-order* finite difference method.

7.2 A General Approach

In the previous section, we derived three different finite difference methods for computing the derivative at a point x_i using the values of the solution at that point and neighbouring points. We have shown that we can create at least two first-order schemes and one second-order scheme. In the current section, we will demonstrate how to generalize this approach to allow us to obtain schemes of any order and for higher-order derivatives.

To start, we will derive a second-order finite difference method to approximate the first derivative using u_i , u_{i-1} , and u_{i-2} . This procedure can be broken down into four basic steps.

7.2.1 Step 1: Generate the Taylor Series

The first step is to expand a Taylor series about each of the points that will be used to approximate the derivative. The Taylor series for u_{i-1} was given before and is

$$u_{i-1} = u_i - \Delta x \left. \frac{\partial u}{\partial x} \right|_{x_i} + \frac{\Delta x^2}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} - \frac{\Delta x^3}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^4). \quad (7.10)$$

Similarly, since the distance between u_i and u_{i-2} is $2\Delta x$ we get the following Taylor series for that point

$$u_{i-2} = u_i - 2\Delta x \left. \frac{\partial u}{\partial x} \right|_{x_i} + \frac{(2\Delta x)^2}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} - \frac{(2\Delta x)^3}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^4). \quad (7.11)$$

7.2.2 Step 2: Rearrange the Taylor Series

The second step is to rearrange each of the Taylor series generated in Step 1 to obtain the derivative of interest on the LHS with everything else on the RHS. Since we are interested in finding an approximation for the first derivative, we rearrange Equation 7.10

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = \frac{u_i - u_{i-1}}{\Delta x} + \frac{\Delta x}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} - \frac{\Delta x^2}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^3), \quad (7.12)$$

and rearrange Equation 7.11

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = \frac{u_i - u_{i-2}}{2\Delta x} + \frac{2\Delta x}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} - \frac{(2\Delta x)^2}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^3), \quad (7.13)$$

which gives two expressions for the first derivative.

7.2.3 Step 3: Determine a Suitable Combination

If we look at the previous two expressions for the first derivative, we will notice that they are both first-order accurate since the leading term that will get truncated is $\mathcal{O}(\Delta x)$. However, we were tasked with finding a second-order accurate scheme. In order to achieve this, we will try to combine Equation 7.17 and Equation 7.13 in such a way the first-order error term cancels out.

We want to combine them in such a way that

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = a(7.17) + b(7.13), \quad (7.14)$$

gives a second-order approximating for the first derivative, and we need to find the coefficients a and b to achieve this. If we look at the left-hand side, we want to keep the first derivative there. This can be achieved if we ensure that

$$a + b = 1. \quad (7.15)$$

The second thing we need to do is cancel out the $\mathcal{O}(\Delta x)$ term to ensure that the leading truncation error term is $\mathcal{O}(\Delta x^2)$. Looking at the $\mathcal{O}(\Delta x)$ terms, in order for them to cancel out, we require

$$\frac{a}{2} + b = 0. \quad (7.16)$$

Equations 7.15 and 7.16 are recognizable as a linear system of equations with two equations and two unknowns. This can be readily solved using substitution yielding $a = 2$ and $b = -1$.

7.2.4 Step 3: Combine the Schemes

Now that we have determined the constants in Equation 7.14, the last step is to just go ahead and add things together. Doing this will yield that following finite difference approximation

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = \frac{3u_i - 4u_{i-1} + u_{i-2}}{2\Delta x} + \mathcal{O}(\Delta x^2), \quad (7.17)$$

which is a second-order approximation of the first derivative using three points. Hence, we have accomplished our task.

7.3 The Second Derivative

In the previous sections, we derived four different schemes for the first derivative, two that were first-order accurate, and two that were second-order accurate. However, if we go back and look at the Navier-Stokes equations, we will note that we also need to approximate second-derivatives for the diffusive terms. We will derive an example scheme here, which also demonstrates another application of the four basic steps of creating a finite difference scheme. Our objective will be to derive a second-order finite difference method to approximate the second derivative using u_i , u_{i-1} , and u_{i+1} .

We now recall that the first step was to simply expand a Taylor series about all of the points that are not u_i . We have already derived these expansions for u_{i-1} and u_{i+1} , which are

$$u_{i-1} = u_i - \Delta x \left. \frac{\partial u}{\partial x} \right|_{x_i} + \frac{\Delta x^2}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} - \frac{\Delta x^3}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^4), \quad (7.18)$$

and

$$u_{i+1} = u_i + \Delta x \left. \frac{\partial u}{\partial x} \right|_{x_i} + \frac{\Delta x^2}{2} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} + \frac{\Delta x^3}{6} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^4). \quad (7.19)$$

So that is step one done.

In step two, we needed to rearrange these equations such that the derivative of interest is on the LHS. This yields two possible expressions for the second derivative by rearranging Equation 7.18

$$\left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} = \frac{2(u_{i-1} - u_i)}{\Delta x^2} + \frac{2}{\Delta x} \left. \frac{\partial u}{\partial x} \right|_{x_i} + \frac{\Delta x}{3} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^2), \quad (7.20)$$

and Equation 7.19

$$\left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} = \frac{2(u_{i+1} - u_i)}{\Delta x^2} - \frac{2}{\Delta x} \left. \frac{\partial u}{\partial x} \right|_{x_i} - \frac{\Delta x}{3} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \mathcal{O}(\Delta x^2), \quad (7.21)$$

noting that the $\mathcal{O}(\Delta x^4)$ terms are now $\mathcal{O}(\Delta x^2)$ since we had to divide the RHS by Δx^2 . That marks the end of step two.

In the third step, we have to determine a linear combination of these two equations that will give us an expression for the second derivative with a truncation error of $\mathcal{O}(\Delta x^2)$. Since we want to keep the second derivative on the LHS, we require

$$a + b = 1. \quad (7.22)$$

And, in order for the truncation error to be of $\mathcal{O}(\Delta x^2)$, we need to cancel the second and third terms on the RHS of the equations, which are $\mathcal{O}(\Delta x^{-1})$ and $\mathcal{O}(\Delta x)$, respectively. Interestingly, cancelling out both of these terms requires

$$a - b = 0. \quad (7.23)$$

Once again, we have a linear system with two equations and two unknowns. Solving this system yields $a = b = 1/2$, which finishes part three.

Finally, in part four, we simply combine the two equations multiplied by their respective a and b coefficients. This yields our second-order accurate expression for the second-derivative as

$$\left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} = \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} + \mathcal{O}(\Delta x^2). \quad (7.24)$$

7.4 Example Applications

Now that we have created a few introductory finite difference methods, it is time to put them to work on our three simplified systems.

7.4.1 Linear Advection

We recall that the one-dimensional linear advection equation in differential form was

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0. \quad (7.25)$$

In the previous sections, we derived a few options for this, but to start we will use the simple first-order upwind scheme to approximate the spatial derivative at each grid point, such that

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = \frac{u_i^t - u_{i-1}^t}{\Delta x} + \mathcal{O}(\Delta x), \quad (7.26)$$

where the superscript t denotes that we are using the solution at the current time. We will also use a first-order finite difference approach in time to approximate the temporal derivative as well. Hence,

$$\left. \frac{\partial u}{\partial t} \right|_{x_i} = \frac{u_i^{t+1} - u_i^t}{\Delta t} + \mathcal{O}(\Delta t), \quad (7.27)$$

where Δt is the time step size and the superscript $t + 1$ is the solution at the next time step.

Now we simply insert these approximations into the linear advection equation, yielding

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} + \alpha \frac{u_i^t - u_{i-1}^t}{\Delta x} + \mathcal{O}(\Delta x, \Delta t) = 0. \quad (7.28)$$

By inspecting this equation, and assuming that we know the solution at each grid point at the current instant in time, we can rearrange this to make a prediction for the solution at the next time step

$$u_i^{t+1} = u_i^t - \frac{\alpha \Delta t}{\Delta x} (u_i^t - u_{i-1}^t) + \mathcal{O}(\Delta x, \Delta t). \quad (7.29)$$

Looking at Equation 7.29, we note that, provided we know the current solution at all grid points at time t , we can predict the solution at time $t + \Delta t$ with an error term of $\mathcal{O}(\Delta x, \Delta t)$. Then we will have an approximation of the solution at all grid points at time $t + \Delta t$, and we can use this to find an approximation of the solution at time $t + 2\Delta t$ by applying the same equation, and so on. Hence, we are able to advance the linear advection equation to any final time we desire, noting that each time step we take will introduce some numerical error. A series of example solutions using different numbers of grid points is shown in Figure 7.2 with an advection velocity of $\alpha = 1$, and a final solution time of $t = 1$, which allows the Gaussian bump to travel once through the domain. If the error is measured in terms of the height of the peak of the Gaussian bump, it is apparent that it converges linearly as the grid spacing is refined.

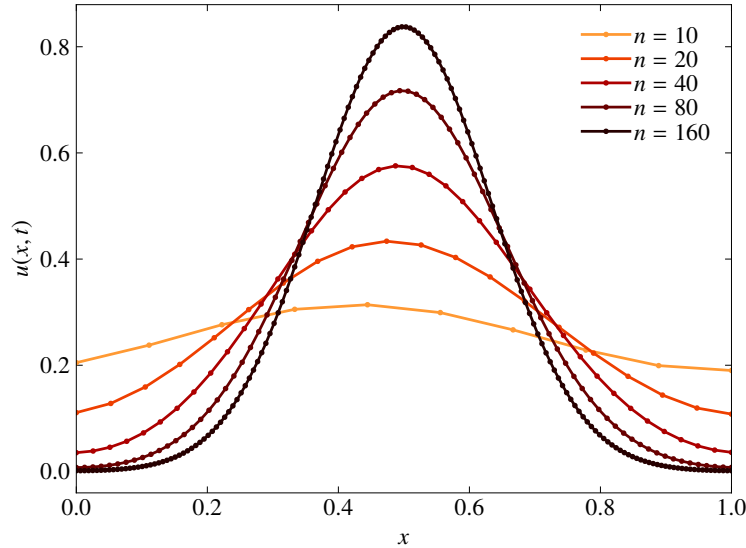


Figure 7.2: First-order advection of a gaussian bump using different levels of grid refinement.



Check out the Advection Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

7.4.2 Burgers Equation

We recall that the one-dimensional Burgers equation was

$$\frac{\partial u}{\partial t} + \frac{1}{2} \frac{\partial u^2}{\partial x} = 0. \quad (7.30)$$

We can now use the exact same approach for Burgers as we used for the linear advection equation. First, we will use a first-order upwind approach to approximate the spatial derivative of u^2 at each grid point x_i , such that

$$\left. \frac{\partial u^2}{\partial x} \right|_{x_i} = \frac{(u_i^t)^2 - (u_{i-1}^t)^2}{\Delta x} + \mathcal{O}(\Delta x), \quad (7.31)$$

where again the superscript t denotes that we are using the solution at the current time. We will also use a first-order finite difference approach in time to approximate the temporal derivative as well. Hence,

$$\left. \frac{\partial u}{\partial t} \right|_{x_i} = \frac{u_i^{t+1} - u_i^t}{\Delta t} + \mathcal{O}(\Delta t). \quad (7.32)$$

Now, we simply insert our finite difference approximations of these two derivatives into the Burgers equation

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} + \frac{1}{2} \frac{(u_i^t)^2 - (u_{i-1}^t)^2}{\Delta x} + \mathcal{O}(\Delta x, \Delta t) = 0, \quad (7.33)$$

and, just like for linear advection, we can now find an expression for the solution at any grid point at the next time step as

$$u_i^{t+1} = u_i^t - \frac{\Delta t}{\Delta x} \frac{1}{2} \left((u_i^t)^2 - (u_{i-1}^t)^2 \right) + \mathcal{O}(\Delta x, \Delta t). \quad (7.34)$$

Looking at this expression, we note it is remarkably similar to the expression for linear advection, which is not particularly surprising as the two initial partial differential equations are also quite similar. Again, if we know the solution at each grid point at some time t , we can approximate the solution at some time $t + \Delta t$ using this expression. Then, to approximate the solution at the time $t + 2\Delta t$ we simply re-apply the expression. This allows us to approximate the behaviour of Burgers equation up to any desired final time, and the error of the approximation is expected to be first-order accurate in both space and time, with errors on the order of $\mathcal{O}(\Delta x, \Delta t)$. A series of example solutions using different numbers of grid points is shown in Figure 7.3.



Check out the Burgers Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

7.4.3 Linear Diffusion

We recall that our third and final simplified system of equations was the linear diffusion equation

$$\frac{\partial u}{\partial t} - \beta \frac{\partial^2 u}{\partial x^2} = 0, \quad (7.35)$$

which described how some conserved quantity, such as heat, diffuses into the surrounding fluid. To approximate this using the finite difference method, we follow the exact same steps as for linear

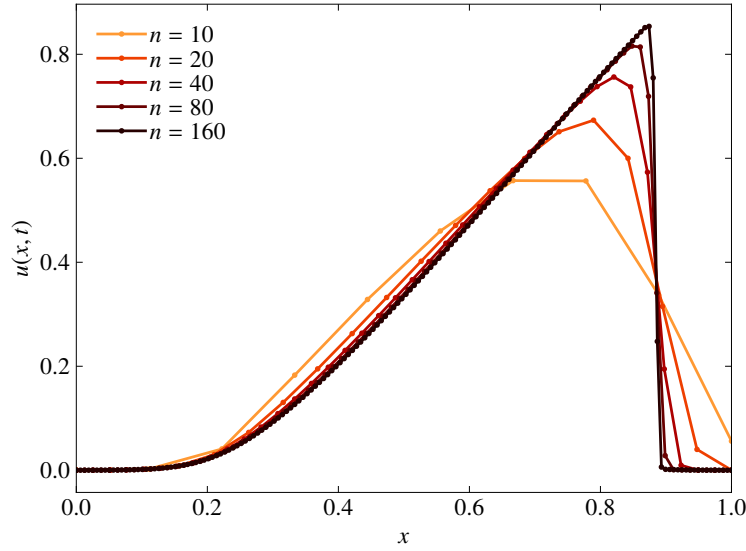


Figure 7.3: First-order nonlinear burgers equation with a gaussian bump using different levels of grid refinement.

advection and Burgers equation. First, we will use our second-order accurate finite difference approximation for the second derivative as

$$\left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} = \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} + \mathcal{O}(\Delta x^2). \quad (7.36)$$

We will also use a first-order finite difference approach in time to approximate the temporal derivative as well. Hence,

$$\left. \frac{\partial u}{\partial t} \right|_{x_i} = \frac{u_i^{t+1} - u_i^t}{\Delta t} + \mathcal{O}(\Delta t). \quad (7.37)$$

Now we simply replace the exact derivatives in the linear diffusion equation with our finite difference approximations, such that

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} - \beta \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} + \mathcal{O}(\Delta x^2, \Delta t) = 0. \quad (7.38)$$

Again, by simply rearranging we can arrive at an expression for the solution at any grid point i at the next time step as

$$u_i^{t+1} = u_i^t + \frac{\beta \Delta t}{\Delta x^2} (u_{i-1} - 2u_i + u_{i+1}) + \mathcal{O}(\Delta x^2, \Delta t). \quad (7.39)$$

Again, this is very similar to the approximations for linear advection and Burgers equations. However, one important thing to note is that the error in space is of $\mathcal{O}(\Delta x^2)$ instead of $\mathcal{O}(\Delta x)$, meaning that as we refine the grid the solution will converge more quickly to the exact solution of the linear diffusion equation.



Check out the Diffusion Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

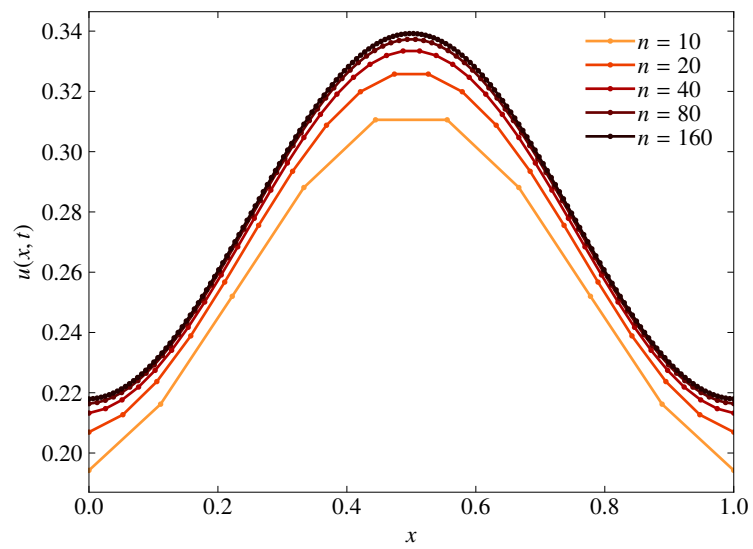


Figure 7.4: Second-order linear diffusion of a gaussian bump using different levels of grid refinement and $\alpha = 1 \times 10^{-3}$.

8. Finite Volume Methods

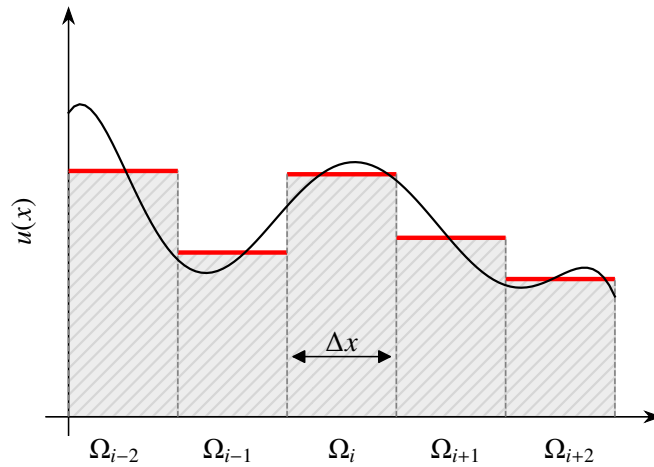


Figure 8.1: One-dimensional Finite Volume discretization

In the previous chapter, we derived and characterized the finite difference method. While it is generally simple to apply, it requires the use of structured grids. Suppose now you want to simulate flow over an aircraft or one of its components. Generating structured meshes can become challenging, if not impossible, for complex geometries. Hence, we need methods that are flexible enough such that the geometry is not a fundamental limitation. Instead of evaluating the solution in a pointwise fashion, consider subdividing a generic domain Ω into volumes or cells Ω_i , as shown in Figure 8.1. Within each cell, the value of the function $u(x)$ can be assumed constant and is calculated using an integral approach. This idea leads to the derivation of the finite volume method, which is one of the most commonly used spatial discretizations in industry. It was initially developed for its application on the general form of the conservation laws, meaning it can be fundamentally applied to solutions that present discontinuities. In this chapter, we present the theory required to apply the finite volume method on scalar as well as systems of conservation laws.

Additional useful references include [1, 7, 13, 22].

8.1 Derivation

Consider the general conservation law applied to a volume of arbitrary shape Ω , as shown in Figure 8.2

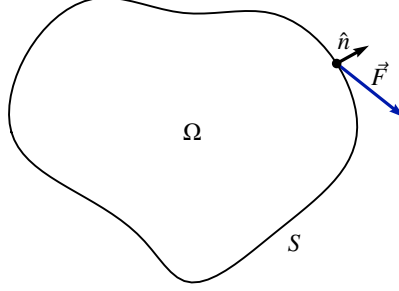


Figure 8.2: Representation of an arbitrary volume of undefined shape.

$$\int_{\Omega} \frac{\partial \vec{u}}{\partial t} d\Omega + \int_S \vec{F} \cdot \hat{n} ds = 0, \quad (8.1)$$

where \vec{u} is a vector of conserved variables, such as mass, momentum, and energy. $\vec{F} = \vec{F}(\vec{u})$ is the vector of fluxes, S is the area of Ω with outward unit normal \hat{n} . Recall that for constant volumes, integration and differentiation commute so we can rewrite Equation 8.1

$$\frac{d}{dt} \int_{\Omega} \vec{u} d\Omega + \int_S \vec{F} \cdot \hat{n} ds = 0. \quad (8.2)$$

Clearly, the integral of the solution represents its total amount within Ω . Hence, the rate of change of the total amount of \vec{u} in a volume is the result of the net flux across the surface S of Ω . As we have stated in the introduction of this chapter, we are looking for averaged values of the solution, specifically the *cell-average* of \vec{u} , which is given by

$$\vec{\bar{u}} = \frac{\int_{\Omega} \vec{u} d\Omega}{|\Omega|}, \quad (8.3)$$

where $|\Omega|$ is the size of the cell. After averaging the solution, it is clear that it has become constant within each volume, i.e. $\vec{\bar{u}} \neq \vec{u}(\vec{x})$, so it is now only a function of time $\vec{\bar{u}} = \vec{\bar{u}}(t)$. If we combine Equations 8.2 and 8.3, we obtain

$$\frac{d\vec{\bar{u}}}{dt} + \frac{1}{|\Omega|} \int_S \vec{F} \cdot \hat{n} ds = 0. \quad (8.4)$$

Now, we have an equation for the rate of change of the averaged quantity $\vec{\bar{u}}$. What remains is to define the value of \vec{F} . First, we note that the averaged solution $\vec{\bar{u}}$ must converge to the instantaneous solution \vec{u} in the limit of sufficiently small volumes

$$\lim_{|\Omega| \rightarrow 0} \vec{\bar{u}} = \vec{u}, \quad (8.5)$$

and hence,

$$\lim_{|\Omega| \rightarrow 0} \vec{F}(\vec{\bar{u}}) = \vec{F}(\vec{u}). \quad (8.6)$$

This leads to the general form of the finite volume method, which we write

$$\frac{d\vec{u}}{dt} = -\frac{1}{|\Omega|} \int_s \vec{F}(\vec{u}) \cdot \hat{n} ds = 0, \quad (8.7)$$

where d/dt can be solved using an appropriate temporal discretization (see Chapter 12).

Generally, we can make relatively good approximations of arbitrary volumes using straight-sided elements such as triangles and tetrahedra. Consider the triangular volume Ω_i displayed in Figure 8.3. The solution \vec{u} is constant within the triangular shape. In addition, each of the faces has an area of S_1 , S_2 and S_3 . Observe that since the sides of the considered volume are straight, the flux vectors are constant across each of the faces. Hence, the surface integral in Equation 8.7 can be trivially solved via summation. For the i -th volume, the general form of the FV method for straight grids can be written

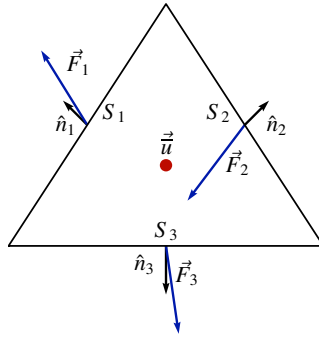


Figure 8.3: Representation of solution, fluxes and faces of a straight-sided element.

$$\frac{d\vec{u}_i}{dt} = -\frac{1}{|\Omega_i|} \sum_{j=1}^m \vec{F}_j \cdot \hat{n}_j S_j, \quad (8.8)$$

where m is the number of faces, F_i is the flux across the j -th face with outward unit normal \hat{n}_j and area surface S_j .

8.2 The Riemann Problem

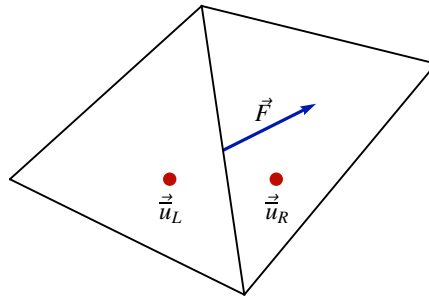


Figure 8.4: Representation of the Riemann problem

You may now realize that the solution is piecewise linear, i.e., we have approximated a function

$\vec{u}(x, t)$ by constant values $\vec{u}_i(t)$ at each volume Ω_i . Usually, in finite difference methods, the value of the flux can be simply computed as $\vec{F} = \vec{F}(\vec{u})$ due to the pointwise approach. In contrast, consider the two volumes shown in Figure 8.4 as a result of a finite volume discretization. We are looking for the value of the fluxes across each of the volume's faces to solve Equation 8.8. However, note that on each side of the interface, we have two values of the solution \vec{u}_L, \vec{u}_R , which are expected to yield a single value of the flux. Hence, the flux at the interface is

$$\vec{F} = \vec{F}(\vec{u}_L, \vec{u}_R). \quad (8.9)$$

This is known as the *Riemann problem*. The solution to the Riemann problem depends on the physics of the problem. For hyperbolic equations, it is said to be a similarity solution and is related to the *characteristic speed* of the PDE. We now present some examples of the Riemann problem for scalar conservation laws and the resulting finite-volume discretizations. Later in this chapter, we will come back to this topic and derive some solutions for systems of hyperbolic equations.

8.3 Example Applications

We have now derived the finite volume method and described some of its characteristics. We have introduced a consequence of the discretization, which is the Riemann problem. In this section, we present finite-volume schemes applied to our simple one-dimensional scalar conservation laws, as well as solutions to the Riemann problem for each of these equations.

8.3.1 Linear Advection

Consider the general conservation law applied to the one-dimensional advection equation. For scalar conservation laws, we may write

$$\vec{u} = u, \quad (8.10)$$

and hence the advection flux is given by

$$\vec{F} = f = \alpha u, \quad (8.11)$$

where α is the advection velocity. A finite-volume discretization for this problem can be seen in Figure 8.5, where $\alpha > 0$. Consider the cell Ω_i . We need to find the value of the flux at each of the faces of the cell. Different approaches can be considered, which result in distinct finite-volume

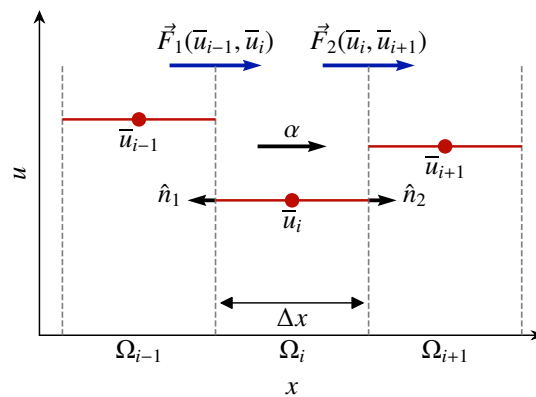


Figure 8.5: Finite-volume linear advection.

schemes, each with its own characteristic error and order of accuracy. We now present three options to compute the Riemann flux and the resulting discretizations.

Upwind

We have previously assumed $\alpha > 0$, which means information travels from left to right. A natural choice of Riemann flux is *upwind*. Here, the value of the flux is computed from the left-hand-side solution value at each interface. Hence,

$$\vec{F}(\bar{u}_L, \bar{u}_R) = \alpha \bar{u}_L, \quad (8.12)$$

which can be written

$$\vec{F}_1 = \alpha \bar{u}_{i-1}, \quad (8.13)$$

$$\vec{F}_2 = \alpha \bar{u}_i, \quad (8.14)$$

for the left and right faces of Ω_i , respectively. Recall the finite-volume discretization for straight-sided elements in Equation 8.8, which we now rewrite for this problem considering $\hat{n}_1 = -1$, $\hat{n}_2 = 1$ and $S_1 = S_2 = 1$

$$\frac{d\bar{u}_i}{dt} = -\frac{1}{\Delta x} (\alpha \bar{u}_i - \alpha \bar{u}_{i-1}), \quad (8.15)$$

or simply

$$\frac{d\bar{u}_i}{dt} = -\alpha \frac{\bar{u}_i - \bar{u}_{i-1}}{\Delta x}, \quad (8.16)$$

which is identical to the original first-order finite-difference method. Discretizing the temporal derivative using the forward Euler method, we have

$$\bar{u}_i^{t+1} = \bar{u}_i^t - \frac{\alpha \Delta t}{\Delta x} (\bar{u}_i^t - \bar{u}_{i-1}^t). \quad (8.17)$$

Hence, by comparing the above equation with Equation 7.29, we can see that they are identical.

Central

A second choice of Riemann flux results from an averaged value of the solution at each interface. This means

$$\vec{F}(\bar{u}_L, \bar{u}_R) = \frac{1}{2} (\alpha \bar{u}_L + \alpha \bar{u}_R). \quad (8.18)$$

Hence, the fluxes can be written

$$\vec{F}_1 = \frac{1}{2} (\alpha \bar{u}_{i-1} + \alpha \bar{u}_i), \quad (8.19)$$

$$\vec{F}_2 = \frac{1}{2} (\alpha \bar{u}_i + \alpha \bar{u}_{i+1}), \quad (8.20)$$

for the left and right interface of Ω_i , respectively. By substituting these fluxes into Equation 8.8, a central finite-volume scheme can be written

$$\frac{d\bar{u}_i}{dt} = -\alpha \frac{\bar{u}_{i+1} - \bar{u}_{i-1}}{2\Delta x}, \quad (8.21)$$

which is identical to the second-order finite-difference method.

Blended

Consider upwind and central fluxes given by

$$f_u = \alpha \bar{u}_L, \quad (8.22)$$

$$f_c = \frac{\alpha}{2} (\bar{u}_L + \bar{u}_R), \quad (8.23)$$

respectively, where f_u yields a *low-resolution* first order scheme and f_c yields a second-order, *high-resolution* scheme. We can combine these two approaches to obtain a *blended* scheme, such that

$$f_b = f_u + \phi f_c, \quad (8.24)$$

where ϕ is a weighting parameter that can recover the upwind, central or yield a blended scheme, i.e.

$$f = \begin{cases} f_u & \text{if } \phi = 0, \\ f_c & \text{if } \phi = 1, \\ f_b & \text{if } 0 < \phi < 1. \end{cases}$$

We have shown that we can recover finite-difference schemes on structured grids for the linear advection equation considering appropriate choices for the Riemann solver. We now explore whether this is also true for the Burgers equation.

8.3.2 Burgers Equation

We now consider the Burgers equation. As we have previously stated, for scalar conservation laws

$$\vec{u} = u, \quad (8.25)$$

and in the case of Burgers, the flux can be written

$$\vec{F} = f(u) = \frac{1}{2} u^2. \quad (8.26)$$

Recall that solutions in Burgers equation tend to develop discontinuities at a finite time even if they are initially smooth. Hence, a common Riemann flux choice for this problem is upwind due to its dissipative effect. We can write the fluxes at each face of Ω_i

$$F_1 = \frac{1}{2} \bar{u}_{i-1}^2, \quad (8.27)$$

$$F_2 = \frac{1}{2} \bar{u}_i^2. \quad (8.28)$$

By following the same procedure as with the advection equation, we obtain the semidiscrete scheme from Equation 8.8,

$$\frac{d\bar{u}_i}{dt} = -\frac{1}{\Delta x} \left(\frac{\bar{u}_i^2}{2} - \frac{\bar{u}_{i-1}^2}{2} \right), \quad (8.29)$$

which we simplify

$$\frac{d\bar{u}_i}{dt} = -\frac{1}{2\Delta x} \left((\bar{u}_i)^2 - (\bar{u}_{i-1})^2 \right). \quad (8.30)$$

Applying the first-order forward Euler scheme to advance the solution in time yields

$$\bar{u}_i^{t+1} = \bar{u}_i^t - \frac{\Delta t}{\Delta x} \frac{1}{2} \left((\bar{u}_i^t)^2 - (\bar{u}_{i-1}^t)^2 \right). \quad (8.31)$$

Equation 8.31 is identical to the first-order upwind scheme for Burgers equation derived using finite difference in Equation 7.34.

8.3.3 Linear Diffusion

Consider the scalar linear diffusion equation, where

$$\vec{u} = u, \quad (8.32)$$

and the corresponding flux can be obtained from Fourier's law

$$\vec{F} = f = -\beta \frac{\partial u}{\partial x}, \quad (8.33)$$

where β is a constant diffusion coefficient. The resulting finite-volume discretization for this problem can be observed in Figure 8.6. Note that at each interface, we need to compute the derivative of the solution. Due to the behaviour of the diffusion equation, one should expect the Riemann flux to include the effects from both sides of the element. We can compute the derivatives

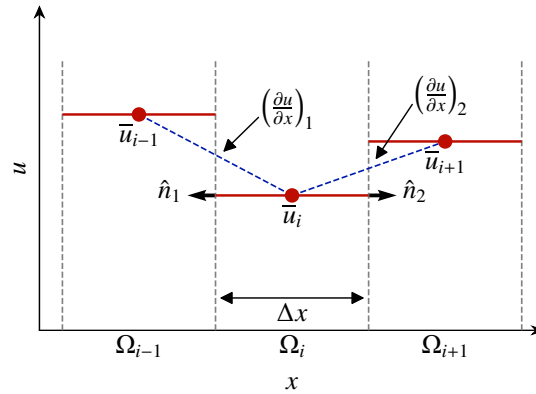


Figure 8.6: Finite-volume linear diffusion.

considering an upwind and a downwind difference approach at the left and right faces, respectively. Hence,

$$\left(\frac{\partial u}{\partial x} \right)_1 \approx \frac{\bar{u}_i - \bar{u}_{i-1}}{\Delta x}, \quad (8.34)$$

$$\left(\frac{\partial u}{\partial x} \right)_2 \approx \frac{\bar{u}_{i+1} - \bar{u}_i}{\Delta x}. \quad (8.35)$$

Then the respective fluxes can be computed from

$$\vec{F}_1 = -\beta \left(\frac{\partial u}{\partial x} \right)_1 \approx -\beta \frac{\bar{u}_i - \bar{u}_{i-1}}{\Delta x}, \quad (8.36)$$

$$\vec{F}_2 = -\beta \left(\frac{\partial u}{\partial x} \right)_2 \approx -\beta \frac{\bar{u}_{i+1} - \bar{u}_i}{\Delta x}. \quad (8.37)$$

By substituting the fluxes into Equation 8.8, we obtain the finite-volume method for linear diffusion

$$\frac{d\bar{u}_i}{dt} = \beta \frac{\bar{u}_{i+1} - 2\bar{u}_i + \bar{u}_{i-1}}{\Delta x^2}, \quad (8.38)$$

which is second-order accurate. The resulting scheme is central, which agrees with the physics of diffusive processes. If we couple the spatial derivative in Equation 8.38 with the forward Euler method, we can write the fully-discrete scheme

$$\bar{u}_i^{t+1} = \bar{u}_i^t + \frac{\beta \Delta t}{\Delta x^2} (\bar{u}_{i+1}^t - 2\bar{u}_i^t + \bar{u}_{i-1}^t). \quad (8.39)$$

Note that Equation 8.38 is identical to the second-order finite-difference scheme we previously derived for this problem in Equation 7.39.

8.4 Linear Hyperbolic Problems

In Chapter 2, we stated that first-order partial differential equations are naturally hyperbolic, meaning that they exhibit wave-like solutions. The simplest case is our advection equation

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad (8.40)$$

which can model, for instance, the tracing of a dye particle in a one-dimensional geometry, such as

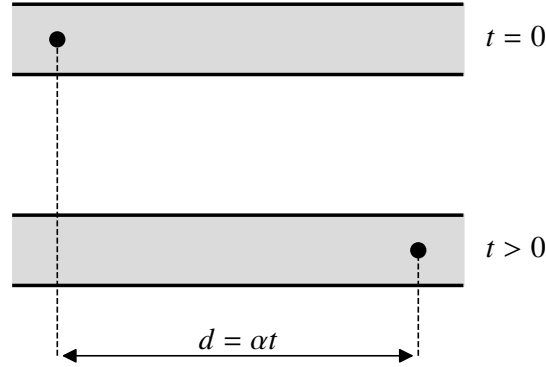


Figure 8.7: Initial and final state of the advection of a dye particle in a one-dimensional flow.

a pipe. Suppose that you want to trace this particle. At time $t = 0$, the particle is located at $x = 0$ as shown in Figure 8.7. After some time, we expect the particle to have translated a distance $d = \alpha t$. Hence, it can be easily checked that solutions to Equation 8.40 are given by

$$u(x, t) = u_0(x - \alpha t), \quad (8.41)$$

where u_0 is the initial solution. Note that the particle has only changed location, and no shape changes or any diffusion effects are modelled by Equation 8.40.

Consider now the space-time plane $x \times t$. In this space, we can find *characteristic lines* $x(t)$ along which the solution remains unchanged, that is, lines where $du = 0$. We can then rewrite the solution as $u(x(t), t)$, and by using the chain rule, we have

$$\frac{d}{dt}(u(x(t), t)) = \frac{\partial u}{\partial t} + \frac{dx}{dt} \frac{\partial u}{\partial x} = 0. \quad (8.42)$$

Now, compare Equations 8.42 and 8.40. It is clear that

$$\frac{dx}{dt} = \alpha. \quad (8.43)$$

Hence the *characteristic speed* of a scalar hyperbolic equation is its advection velocity. From the above relation, we can obtain an equation for the characteristic lines of our PDE, given by

$$x = x_0 + \alpha t, \quad (8.44)$$

where a curve can be drawn for each initial value of x_0 corresponding to $u(x, t = 0) = u(x_0)$.

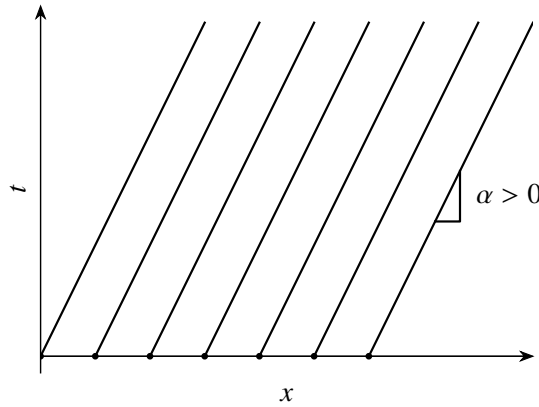


Figure 8.8: Characteristic lines for the linear advection equation with $\alpha > 0$ for different values of x_0 .

Consider α to be positive. The resulting characteristic curves are shown in Figure 8.8. Observe that the slope of the curves is α . Hence, we can find the value of u at any point (x, t) by tracing curves parallel to those in Figure 8.8. We have characterized the behaviour of scalar hyperbolic equations. To analyze systems of equations, we must take into account additional considerations, which we explore in the next section.

8.4.1 Linear Hyperbolic Systems

The advective components in our governing equations are responsible for the propagation of information from one point to another. In general, scalar conservation laws are simplifications of more complicated systems. Some of these systems are hyperbolic, which we can analyze as a superposition of advection equations with their corresponding wave speed. A system of m equations

$$\frac{\partial \vec{u}}{\partial t} + A \frac{\partial \vec{u}}{\partial x} = 0, \quad \vec{u} = [u_1 \quad u_2 \quad \dots \quad u_m], \quad (8.45)$$

is said to be hyperbolic if the constant $m \times m$ matrix A , which is generally dense, has m real distinct eigenvalues λ_i and m linearly independent eigenvectors \vec{q}_i . This can be written as the eigenvalue problem

$$A \vec{q}_i = \lambda_i A, \quad i = 1, 2, \dots, m, \quad (8.46)$$

or in matrix form

$$AQ = Q\Lambda, \quad (8.47)$$

where Q is the matrix of eigenvectors

$$Q = [\vec{q}_1 \quad \vec{q}_2 \quad \dots \quad \vec{q}_m], \quad (8.48)$$

and Λ is the diagonal matrix containing the eigenvalues of A

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \lambda_m \end{bmatrix}. \quad (8.49)$$

We have shown in the previous section that the scalar linear equation simply propagates information at a given characteristic speed, which represents the slope of the characteristic curves $x(t)$ in a space-time plane. In the case of a linear hyperbolic system, more than a single wave speed is embedded in the equations. Hence, multiple slopes are associated with any given (x, t) point in the $x \times t$ plane, which are given by the eigenvalues of A . Now, how are these characteristic speeds related to \vec{u} ? It turns out that the solution is considered to be a superposition of these waves. To illustrate this, the original system needs to be decoupled into m independent equations. This is done by diagonalizing the constant matrix A .

Recall the m eigenvectors of A are linearly independent. Hence, we can transform Equation 8.45 into *characteristic form* by setting

$$A = Q\Lambda Q^{-1}. \quad (8.50)$$

Now, substituting A in Equation 8.45 using 8.50 and multiplying by Q^{-1} yields

$$Q^{-1} \frac{\partial \vec{u}}{\partial t} + Q^{-1} Q \Lambda Q^{-1} \frac{\partial \vec{u}}{\partial x} = 0. \quad (8.51)$$

For simplicity, define the vector of *characteristic variables* $\vec{\omega} = Q^{-1} \vec{u}$. Hence, the characteristic form of hyperbolic system of PDEs is

$$\frac{\partial \vec{\omega}}{\partial t} + \Lambda \frac{\partial \vec{\omega}}{\partial x} = 0. \quad (8.52)$$

We have now decoupled our original system into m independent advection equations of the form

$$\frac{\partial \omega_i}{\partial t} + \lambda_i \frac{\partial \omega_i}{\partial x} = 0, \quad i = 1, \dots, m, \quad (8.53)$$

each of which has solution

$$\omega_i = \omega_i^0(x - \lambda_i t), \quad (8.54)$$

where the initial characteristic state can be found directly from the initial conditions by

$$\vec{\omega}_0 = Q^{-1} \vec{u}_0. \quad (8.55)$$

Similar to the scalar case, we can now draw characteristic curves in the $x \times t$ plane. These can be observed in Figure 8.9, where we have considered the eigenvalues to be sorted

$$\lambda_1 < \lambda_2 < \dots < \lambda_m. \quad (8.56)$$

We have now decomposed the vector of conserved quantities \vec{u} into characteristic variables $\vec{\omega}$ employing the eigenvectors \vec{q}_i . Hence, the original variables can be recovered by coefficients ω_i and their corresponding eigenvector.

$$u(x, t) = \sum_{i=1}^m \omega_i(x, t) \vec{q}_i = \sum_{i=1}^m \omega_i^0(x - \lambda_i t) \vec{q}_i. \quad (8.57)$$

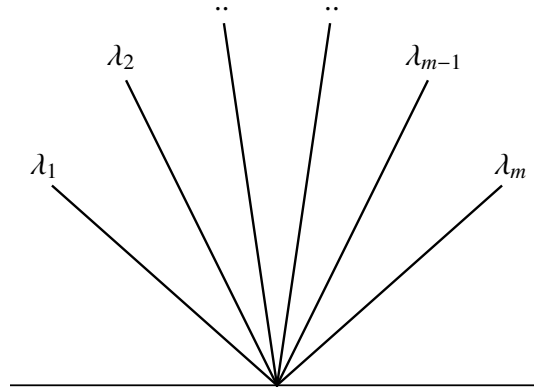


Figure 8.9: Characteristic lines for a linear hyperbolic system of equations.

8.4.2 The Riemann Problem

The Scalar Case

Consider the following initial-value problem (Figure 8.10) for the scalar advection equation

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad u(x, 0) = \begin{cases} u_L & \text{if } x < 0, \\ u_R & \text{if } x > 0, \end{cases} \quad (8.58)$$

where u_L and u_R are piecewise constants and $u_L \neq u_R$. Based on our previous discussions, these

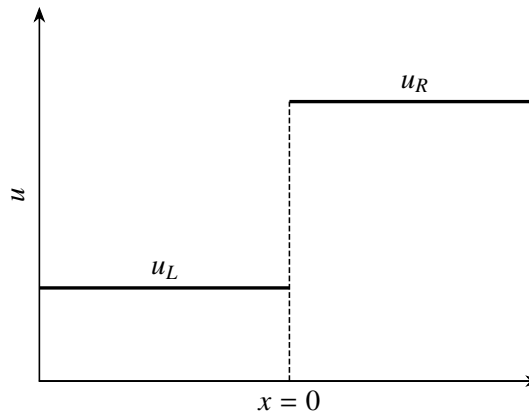


Figure 8.10: Scalar initial condition with discontinuity

constant values simply propagate along some characteristic curve in the $x \times t$ plane along with the discontinuity initially located at $x = 0$. The characteristic line with the origin at this point defines the path of the discontinuity with equation $x = \alpha t$, as shown in Figure 8.11. Hence, on the left side of this curve, specifically where $x < \alpha t$, the solution is simply u_L , and on the right-hand side, where $x > \alpha t$, the solution is u_R . Hence, the Riemann solution to the scalar problem is simply

$$u(x, t) = \begin{cases} u_L & \text{if } x < \alpha t, \\ u_R & \text{if } x > \alpha t. \end{cases} \quad (8.59)$$

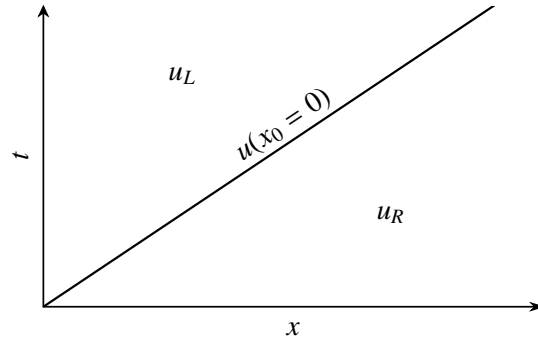


Figure 8.11: Characteristic line along which discontinuity propagates

Hyperbolic Systems

Now consider the following initial-value problem, this time for a system of m hyperbolic equations, where multiple values of the solution exist on each side of $x = 0$

$$\frac{\partial \vec{u}}{\partial t} + A \frac{\partial \vec{u}}{\partial x} = 0, \quad \vec{u}(x, 0) = \begin{cases} \vec{u}_L & \text{if } x < 0, \\ \vec{u}_R & \text{if } x > 0. \end{cases} \quad (8.60)$$

Recall from Section 8.4.1 that linear hyperbolic systems are driven by a superposition of multiple waves. To reveal these underlying advection equations, we must transform the original equations into characteristic form. From Equation 8.55, we may write

$$\vec{\omega}(x, 0) = \begin{cases} \vec{\omega}_L & \text{if } x < 0, \\ \vec{\omega}_R & \text{if } x > 0, \end{cases} \quad (8.61)$$

where the i -th equation has solution

$$\omega_i(x, t) = \begin{cases} \omega_{L,i} & \text{if } x < \lambda_i t, \\ \omega_{R,i} & \text{if } x > \lambda_i t. \end{cases} \quad (8.62)$$

The eigenvalues of A can be observed in Figure 8.12 for the case of $m = 4$ equations. Now, suppose you have a set of coordinates (x, t) where you want to know the value of the solution. At any point in the $x \times t$ domain, the solution only depends on $u(x, 0) = u(x_0) = u_0$. Hence, we can determine the value of \vec{u} at any point by drawing lines parallel to the characteristic curves. On the left of λ_1 , tracing parallel lines only yields initial values which correspond to the left state \vec{u}_L , hence the solution can be computed by

$$\vec{u}_L = \sum_{i=1}^m \omega_{L,i} \vec{q}_i. \quad (8.63)$$

Similarly, at any point on the right side of λ_m (λ_4 in Figure 8.12) we find

$$\vec{u}_R = \sum_{i=1}^m \omega_{R,i} \vec{q}_i. \quad (8.64)$$

Now, what if we are interested in finding the solution at the point P_m in Figure 8.12?. Tracking the solution back to its initial value shows that values of $x - \lambda t$ are located on both the left and right-hand sides of $x = 0$. Hence, we require a combination of both left and right states. Note that at any point within the region defined by λ_2 and λ_3 , the solution \vec{u}_m depends on the linear

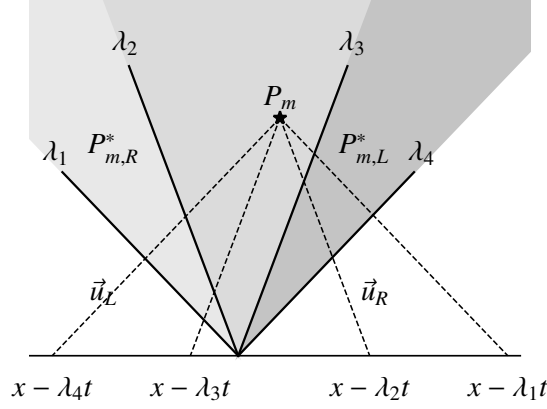


Figure 8.12: Characteristic curves for the Riemann problem with $m = 4$. Note the value of the maximum sub-index changes for each point P . For $P_{m,L}^*$, $I = 1$, for P_m , $I = 2$ and for $P_{m,R}^*$, $I = 3$. (Adapted from [13])

combination of left and right states. Moving P_m to the left or right of these characteristic lines, for example where points $P_{m,L}^*$ and $P_{m,R}^*$ are will require a different combination $\vec{\omega}_L$ and $\vec{\omega}_R$. Hence, to generalize the equation for the solution at any point, define $I^* = I^*(x, t)$ to be the maximum index i for which $x > \lambda_i t$. Then, the solution can be found by

$$\vec{u}_m = \sum_{i=I^*+1}^m \omega_{L,i} \vec{q}_i + \sum_{i=1}^{I^*} \omega_{R,i} \vec{q}_i. \quad (8.65)$$

Since the solution is constant along each characteristic line, the jump between two states is also constant and is given for the whole system

$$\vec{\delta} = Q^{-1}(\vec{u}_R - \vec{u}_L). \quad (8.66)$$

Hence the jump across the λ_i -characteristic is given by $\delta_i = \omega_{R,i} - \omega_{L,i}$, which is known as the strength of the i -th wave. Note that Equation 8.66 is called the *Rankine-Hugoniot* condition, and will be useful for the analysis of nonlinear problems later in this chapter.

Example case: Linear Acoustics

Consider the propagation of sound waves in a motionless gas. Small changes in pressure and density propagate in the surrounding air. This propagation is governed in one dimension by

$$\begin{bmatrix} \rho \\ \rho v \end{bmatrix}_t + \begin{bmatrix} \rho v \\ \rho v^2 + p(\rho) \end{bmatrix}_x = 0. \quad (8.67)$$

These are the Euler equations, one of which is the nonlinear PDE for the conservation of momentum. This system can be written in *quasilinear* form [22]

$$\frac{\partial \vec{u}}{\partial t} + \vec{f}'(\vec{u}) \frac{\partial \vec{u}}{\partial x} = 0, \quad (8.68)$$

where

$$\vec{u} = \begin{bmatrix} \rho \\ \rho v \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \quad (8.69)$$

$$\vec{F}(\vec{u}) = \begin{bmatrix} \rho v \\ \rho v^2 + p(\rho) \end{bmatrix} = \begin{bmatrix} u_2 \\ \frac{u_2^2}{u_1^2} + p(u_1) \end{bmatrix}. \quad (8.70)$$

Then $A = \vec{f}'(\vec{u})$ is the Jacobian matrix and is given by

$$A = \vec{f}'(\vec{u}) = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{u_2^2}{u_1^2} + p'(u_1) & \frac{2u_2}{u_1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -v^2 + p'(\rho) & 2v \end{bmatrix}. \quad (8.71)$$

We can analyze systems with small disturbances by linearizing about some state. This can be done by considering the flow properties to be the sum of an initial variable plus a perturbation term

$$u = \tilde{u}(x, t), \quad (8.72)$$

$$\rho = \rho_0 + \tilde{\rho}(x, t), \quad (8.73)$$

$$p = p_0 + \tilde{p}(x, t), \quad (8.74)$$

and since the gas is considered motionless, we have set $u_0 = 0$. Neglecting products of small disturbances and applying the chain rule to the momentum equations, the linear system can be written

$$\begin{bmatrix} \tilde{\rho} \\ \tilde{u} \end{bmatrix}_t + \begin{bmatrix} 0 & \rho_0 \\ \frac{c^2}{\rho_0} & 0 \end{bmatrix} \begin{bmatrix} \tilde{\rho} \\ \tilde{u} \end{bmatrix}_x = 0, \quad (8.75)$$

where $c = \sqrt{p'(\rho_0)}$ is the speed of sound. Now, consider the following Riemann problem

$$\vec{u}(x, 0) = \vec{u}_0 = \begin{cases} \vec{u}_L & \text{if } x < 0, \\ \vec{u}_R & \text{if } x > 0, \end{cases} \quad (8.76)$$

where $\vec{u}_L = [\rho_L \ u_L]^T$ and $\vec{u}_R = [\rho_R \ u_R]^T$, $\vec{u}_L \neq \vec{u}_R$. Note that the problem is linear due to the above procedure. We can compute the eigenvalues of A in Equation 8.75 by setting

$$\det(A - I\lambda) = 0, \quad (8.77)$$

where I is the identity matrix. Solving the eigenvalue problem yields the characteristic polynomial

$$\lambda^2 - c^2 = 0, \quad (8.78)$$

which has solutions

$$\lambda_1 = -c, \quad \lambda_2 = +c. \quad (8.79)$$

The corresponding matrix of eigenvectors can be found to be

$$Q = \begin{bmatrix} \rho_0 & \rho_0 \\ -c & c \end{bmatrix}, \quad (8.80)$$

with inverse

$$Q^{-1} = \frac{1}{2\rho_0 c} \begin{bmatrix} c & -\rho_0 \\ c & \rho_0 \end{bmatrix}. \quad (8.81)$$

As we expected, the solution consists of a superposition of left-going and right-going sound waves, which represent the slopes of the characteristic lines shown in Figure 8.13. Now, we analyze the characteristic form of Equation 8.75 seeking a Riemann solution \vec{u}_m . Note that for systems of $m = 2$ equations, the Riemann state only depends on the single region highlighted in Figure 8.13. We start by finding the values of $\vec{\omega}_L$ and $\vec{\omega}_R$ by

$$\vec{\omega}_L = \begin{bmatrix} \omega_{L,1} \\ \omega_{L,2} \end{bmatrix} = Q^{-1} \vec{v}_L = \frac{1}{2c\rho_0} \begin{bmatrix} c & -\rho_0 \\ c & \rho_0 \end{bmatrix} \begin{bmatrix} \rho_L \\ v_L \end{bmatrix}. \quad (8.82)$$

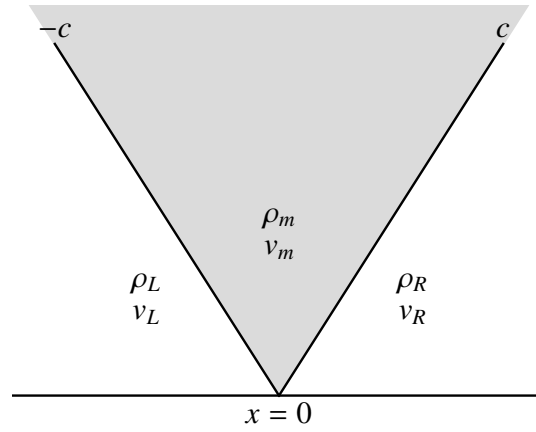


Figure 8.13: Characteristic curves for the linear acoustics example

Hence

$$\omega_{L,1} = \frac{c\rho_L - \rho_0 v_L}{2c\rho_0}, \quad (8.83)$$

$$\omega_{L,2} = \frac{c\rho_L + \rho_0 v_L}{2c\rho_0}. \quad (8.84)$$

Similarly, for the right state we can find

$$\omega_{R,1} = \frac{c\rho_R - \rho_0 v_R}{2c\rho_0}, \quad (8.85)$$

$$\omega_{R,2} = \frac{c\rho_R + \rho_0 v_R}{2c\rho_0}. \quad (8.86)$$

In Equation 8.65, $I^* = 1$, hence

$$\vec{u}_m = \begin{bmatrix} \rho_m \\ v_m \end{bmatrix} = \sum_{i=2}^2 \omega_{L,i} \vec{q}_i + \sum_{i=1}^1 \omega_{R,i} \vec{q}_i = \omega_{L,2} \vec{q}_2 + \omega_{R,1} \vec{q}_1, \quad (8.87)$$

which yields the exact Riemann solution for the linear acoustics equation

$$\rho_m = \frac{1}{2}(\rho_R + \rho_L) + \frac{1}{2} \frac{\rho_0}{c} (u_L - u_R), \quad (8.88)$$

$$u_m = \frac{1}{2} \frac{c}{\rho_0} (\rho_L - \rho_R) + \frac{1}{2} (u_R + u_L). \quad (8.89)$$

Similar to [22], we can define the values of $\rho_0 = 1$, $c = 1$ and given the initial left and right states

$$\vec{u}_L = \begin{bmatrix} \rho_L \\ v_L \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}, \quad \vec{u}_R = \begin{bmatrix} \rho_R \\ v_R \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.0 \end{bmatrix}, \quad (8.90)$$

as shown in Figure 8.14a, we can determine the middle state from Equation 8.89. Hence $\rho_m = 0.375$ and $v_m = 0.125$. Results can be seen in Figure 8.14b, where a symmetric wave can be seen to have moved from $x = 0$ to $-ct$ and ct for the left and right discontinuities, respectively.

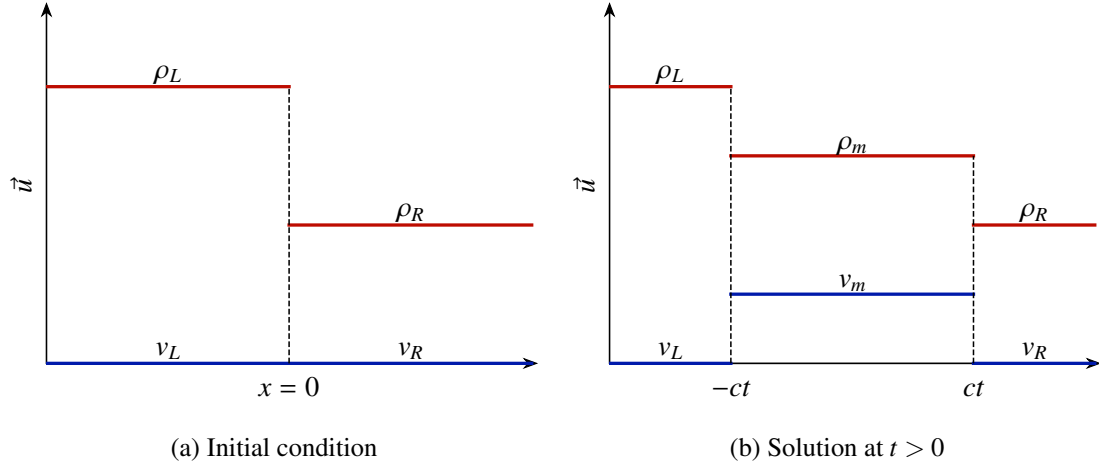


Figure 8.14: Example solution with values for the linear acoustics case.

8.5 Nonlinear Hyperbolic Problems

Consider the following one-dimensional scalar conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0, \quad (8.91)$$

where $f(u)$ is the flux. In Section 8.4, we analyzed a flux function of the form $f(u) = \alpha u$, corresponding to linear advection. In this section, we consider a nonlinear flux function of the form $f(u) = \frac{1}{2}u^2$, which leads to the inviscid Burgers equation. For this type of problem, u is expected not only to translate but also to deform towards the formation of shocks. Similar to Section 8.4, we want to analyze the behaviour of this PDE via the method of characteristics. Hence, we write the *quasilinear form* of Burgers' equation

$$\frac{\partial u}{\partial t} + f'(u) \frac{\partial u}{\partial x} = 0, \quad (8.92)$$

where $f'(u) = u$. Recall that in the $x-t$ plane, we look for lines along which the solution remains unchanged. Using the chain rule for $u(x(t), t)$, we obtain

$$\frac{du}{dt} = \frac{\partial u}{\partial t} + \frac{dx}{dt} \frac{\partial u}{\partial x} = 0. \quad (8.93)$$

Furthermore, comparing Equations 8.93 and 8.92, we observe that

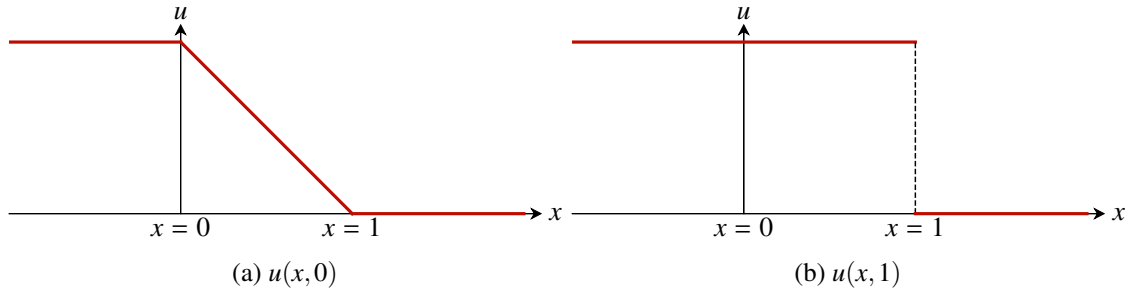
$$\frac{dx}{dt} = u, \quad (8.94)$$

which allows us to obtain an equation for the characteristic line that passes through x_0

$$x = x_0 + u_0 t, \quad (8.95)$$

where $u_0 = u(x, 0)$ is the slope of each line. This means that each characteristic line has an inclination that depends on the initial value of the solution. Note that since u is the characteristic speed of the problem and $f(u)$ is an increasing function, i.e. higher values of the solution travel faster than smaller values of u , causing the solution to deform. To illustrate this, consider the following example shown in Figure 8.15a

$$u(x, 0) = \begin{cases} 1 & \text{if } x < 0, \\ 1 - x & \text{if } 0 < x < 1, \\ 0 & \text{if } x > 1. \end{cases} \quad (8.96)$$

Figure 8.15: Solution at $t = 0$ and $t = 1$ for Equation 8.96

Since the solution is its own convective speed, we expect the region $x < 1$ to move towards the right, and the region after this point to remain stationary. Then a portion of the initial solution will meet at $x = 1$, where a shock will form. Let us now visualize this via the method of characteristics. Using Equation 8.95 at different x -locations, we draw the characteristic lines in Figure 8.16. Multiple curves can be seen to intersect for a single value of u at $(x, t) = (1, 1)$, corresponding to the solution illustrated in Figure 8.15b at $t = 1$.

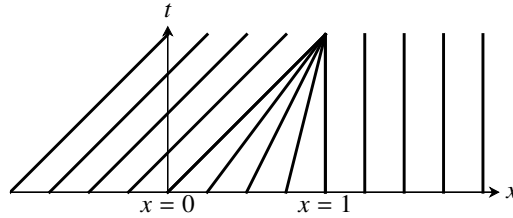


Figure 8.16: Characteristic lines for Equation 8.96.

At $t = 1$, the method of characteristics breaks. To ensure conservation after this moment, the discontinuity is expected only to translate at a speed s . This speed depends on the values of both sides of the discontinuity and can be obtained via the Rankine-Hugoniot relation, which we now derive.

Consider the integral form of the conservation law applied to a finite region in the (x, t) plane where the shock has already formed and only translates, similar to Figure 8.11. This results in

$$\frac{\partial}{\partial t} \int_{x_L}^{x_R} u(x, t) dx + [f(u_R) - f(u_L)] = 0. \quad (8.97)$$

Assuming constant values of u on the left and right side of the discontinuity, we can rewrite Equation 8.97

$$\frac{u(x_R, t) - u(x_L, t)}{\Delta t} \Delta x + f(u_R) - f(u_L) = 0, \quad (8.98)$$

which in the limit of $\Delta t \rightarrow 0$ becomes

$$s = \frac{dx}{dt} = \frac{f(u_R) - f(u_L)}{u_R - u_L}, \quad (8.99)$$

where s is the shock speed. This equation is known as the Rankine-Hugoniot relation. For the Burgers equation with $f(u) = u^2/2$, we find that the shock speed is given by

$$s = \frac{1}{2} \frac{(u_R^2 - u_L^2)}{u_R - u_L} = \frac{1}{2} (u_L + u_R), \quad (8.100)$$

which can be computed for the problem in Equation 8.96 to be $s = \frac{1}{2} (0 + 1) = \frac{1}{2}$.

Weak solutions

Now, consider the following initial conditions

$$u(x, 0) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x > 0. \end{cases} \quad (8.101)$$

In this case, the characteristic lines do not intersect and are displayed in Figure 8.17. While these

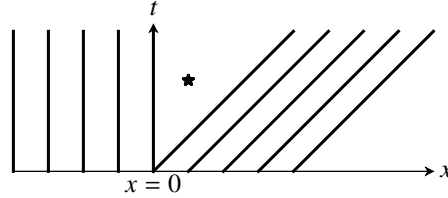


Figure 8.17: Characteristic lines for Equation 8.101.

curves do not cross, we are missing information in the star region. One possible solution of this problem can be written

$$u(x, t) = \begin{cases} 0 & \text{if } x < t/2, \\ 1 & \text{if } x > t/2, \end{cases} \quad (8.102)$$

as illustrated in Figure 8.18a. This form of the solution satisfies the Rankine-Hugoniot relation and is known as an expansion shock, where information appears to come out of a discontinuity. From a physical point of view, this violates the entropy condition. In a physical shock wave, characteristics are expected to go into the discontinuity as time advances. Hence, while Equation 8.103 is a mathematical solution, it is not an entropy or physically relevant solution.

Consider instead

$$u(x, t) = \begin{cases} 0 & \text{if } x \leq t/2, \\ \frac{x}{t} & \text{if } 0 \leq x \leq t, \\ 1 & \text{if } x \geq t, \end{cases} \quad (8.103)$$

which is known as a rarefaction wave, shown in Figure 8.18b. In this case, the solution consists of a smooth transition between the left and right states, consistent with the entropy condition.

Even with smooth initial conditions, nonlinear equations can develop discontinuities. For a general conservation law, we say a solution that satisfies a PDE of order k is a classical solution if the first k derivatives exist and are continuous. So, how is a discontinuous solution admitted in a PDE? It turns out that we need to expand the range of admissible solutions to include those that may not be differentiable so that we can deal with the formation of shocks. The set of both

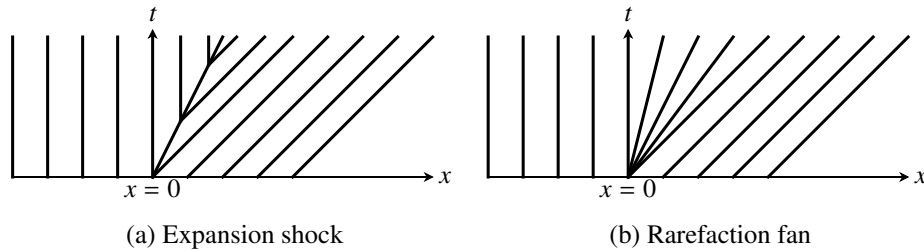


Figure 8.18: Mathematically admissible solutions to problem in Equation 8.101

classical and discontinuous u is generally known as weak solutions. They are considered to satisfy the conservation law in an integral sense, which was the approach that we used to derive the Rankine-Hugoniot relation.

Recall our second example in Equation 8.101. We showed that weak solutions are not necessarily unique, and hence we must add a constraint to single out those relevant in the physical world. Hence, to identify physical solutions, we consider a discontinuity to be a physical shock wave only if the following entropy condition is satisfied

$$f'(u_L) > s > f'(u_R), \quad (8.104)$$

which is consistent with the choice of Equation 8.103 as the physically relevant solution in our second example.

8.5.1 Nonlinear Hyperbolic Systems

Applications of engineering interest can be described by a system of conservation laws of the form

$$\frac{\partial \vec{u}}{\partial t} + \frac{\partial \vec{F}(\vec{u})}{\partial x} = 0, \quad (8.105)$$

where \vec{u} is the vector of conserved variables and $\vec{F}(\vec{u})$ is the vector of fluxes. Previously, we considered constant Jacobian matrices that led to a set of linear equations. In this section, we consider problems whose Jacobian matrices have the form

$$A(\vec{u}) = \frac{\partial \vec{F}(\vec{u})}{\partial \vec{u}}, \quad (8.106)$$

and hence both the eigenvalues and eigenvectors depend on the value of the solution, which in turn depends on time. An example of a nonlinear hyperbolic system is given by the Euler equations, which can be written in one dimension using Equation 8.105 with

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} \rho \\ \rho v \\ \rho E \end{bmatrix}, \quad \vec{F} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} \rho v \\ \rho v^2 + P \\ v(\rho E + p) \end{bmatrix}. \quad (8.107)$$

ρ is density, v is velocity and E is the specific energy

$$E = e + \frac{1}{2}v^2. \quad (8.108)$$

We assume ideal gas law, hence the specific internal energy e is given by

$$e = \frac{P}{(\gamma - 1)\rho}, \quad (8.109)$$

where γ is the ratio of specific heats. In quasilinear form, we write

$$\frac{\partial \vec{u}}{\partial t} + A(\vec{u}) \frac{\partial \vec{u}}{\partial x} = 0, \quad (8.110)$$

where $A(\vec{u})$ is the Jacobian matrix given by

$$A(\vec{u}) = \frac{\partial \vec{F}(\vec{u})}{\partial \vec{u}} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \frac{\partial f_1}{\partial u_3} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \frac{\partial f_2}{\partial u_3} \\ \frac{\partial f_3}{\partial u_1} & \frac{\partial f_3}{\partial u_2} & \frac{\partial f_3}{\partial u_3} \end{bmatrix}, \quad (8.111)$$

To compute $A(\vec{u})$, we write the Euler fluxes in terms of u_1, u_2, u_3

$$\vec{F} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} u_2 \\ u_2^2/u_1 + (\gamma-1)(u_3 - u_2^2/2u_1) \\ u_3 \frac{u_2}{u_1} \gamma - \frac{u_2^3(\gamma-1)}{u_1^2 2} \end{bmatrix}. \quad (8.112)$$

Hence, the Jacobian matrix for the Euler equations is given by

$$A(\vec{u}) = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{2}(3-\gamma)\left(\frac{u_2}{u_1}\right)^2 & (3-\gamma)\left(\frac{u_2}{u_1}\right) & \gamma-1 \\ (\gamma-1)\left(\frac{u_2}{u_1}\right)^3 - \gamma \frac{u_2 u_3}{u_1^2} & \gamma \frac{u_3}{u_1} - \frac{3}{2}(\gamma-1)\left(\frac{u_2}{u_1}\right)^2 & \gamma \frac{u_2}{u_1} \end{bmatrix}, \quad (8.113)$$

or using the conserved variables

$$A(\vec{u}) = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{2}(\gamma-3)v^2 & (3-\gamma)v & \gamma-1 \\ \frac{1}{2}(\gamma-2)v^3 - c^2 \frac{v}{\gamma-1} & \frac{3-2\gamma}{2}v^2 + \frac{c^2}{\gamma-1} & \gamma v \end{bmatrix}, \quad (8.114)$$

where c is the speed of sound

$$c = \sqrt{\frac{\gamma p}{\rho}}. \quad (8.115)$$

The eigenvalues of $A(\vec{u})$ are

$$\lambda_1 = v - c, \quad \lambda_2 = v, \quad \lambda_3 = v + c, \quad (8.116)$$

and the associated eigenvectors are

$$\vec{q}_1 = \begin{bmatrix} 1 \\ v - c \\ H - vc \end{bmatrix}, \quad \vec{q}_2 = \begin{bmatrix} 1 \\ v \\ \frac{v^2}{2} \end{bmatrix}, \quad \vec{q}_3 = \begin{bmatrix} 1 \\ v + c \\ H + vc \end{bmatrix}, \quad (8.117)$$

where H is the total enthalpy defined as

$$H = E + \frac{p}{\rho}. \quad (8.118)$$

The eigenvalues of the Euler system are distinct, and a complete set of eigenvectors can be found, proving that the Euler equations are strictly hyperbolic. The eigenvalues represent the characteristic speeds of the problem, showing that the Euler equations can be interpreted as a superposition of a left-travelling sound wave, a momentum wave and a right-going sound wave, consistent with the propagation of sound. The nonlinearity of these equations makes the study of the Riemann problem much more challenging compared to the linear systems. In the next section, we briefly discuss some important considerations when analyzing the Riemann problem for the Euler equations.

8.5.2 The Riemann Problem for the Euler equations

Consider a tube with a thin membrane at $x = 0$ separating two different gases, each with a given density ρ velocity v , and pressure p as shown in Figure 8.19. Initially, both gases are considered at rest ($v_L = v_R = 0$). Note it can also be the same gas at different pressures. At time t , the thin membrane is ruptured, causing the fluid in the region of high pressure to move towards the

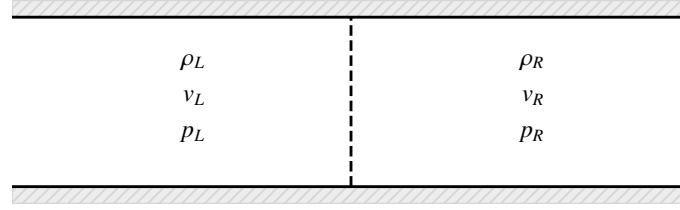


Figure 8.19: Sod's shock-tube problem

low-pressure side to reach thermodynamic equilibrium. This configuration is known as *Sod's shock tube* and is the physical equivalent to the Riemann problem in gas dynamics [22].

It can be shown that the solution to this problem consists of a contact discontinuity associated with λ_2 , and a shock or a rarefaction wave associated with λ_1 and λ_3 . Four different configurations are then possible solutions for the Riemann problem [22], all of which need to be considered when deriving an exact solution. These are shown in Figure 8.20.

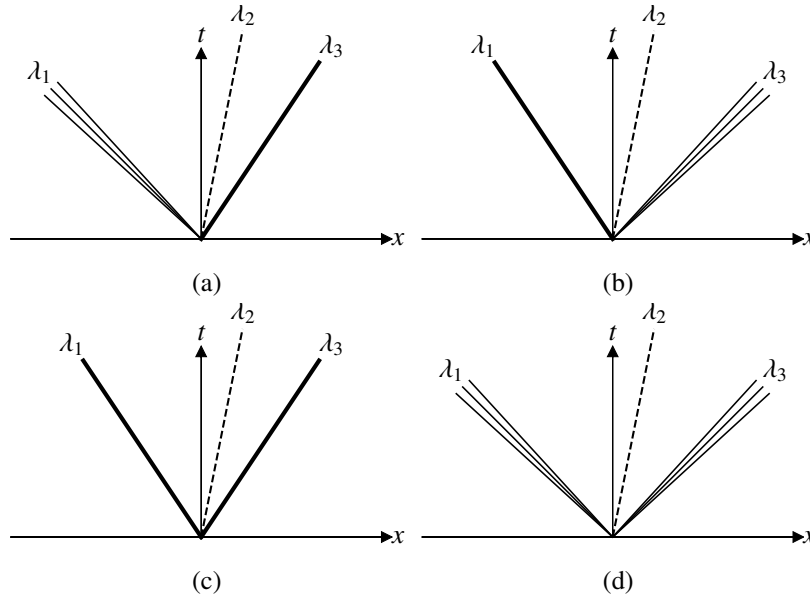


Figure 8.20: Possible characteristic configurations for the Euler Riemann problem

Solving the exact Riemann problem for the Euler equations turns out to be very expensive and requires the use of iterative methods to find the pressure between the left and right states. We refer the reader to [13, 22] for a detailed derivation of the exact solution.

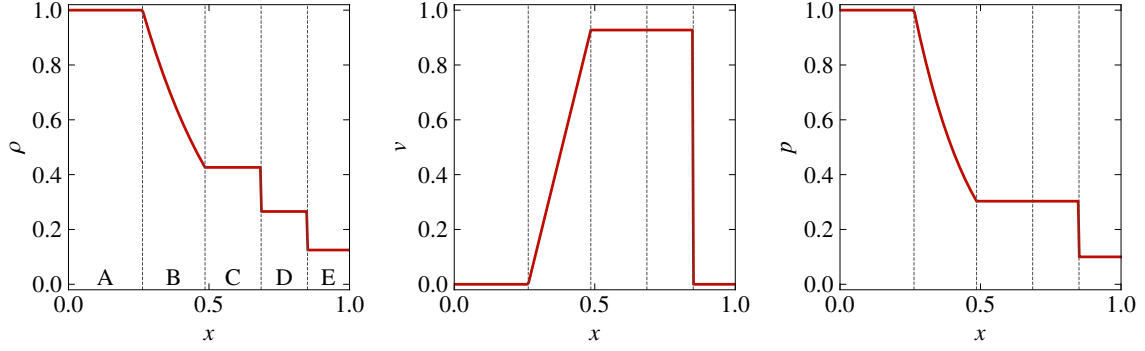
Example

To visualize the behaviour of the Euler equations, consider Sod's shock tube problem with initial conditions

$$\vec{u}_L = \begin{bmatrix} \rho_L \\ v_L \\ p_L \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad \text{and} \quad \vec{u}_R = \begin{bmatrix} \rho_R \\ v_R \\ p_R \end{bmatrix} = \begin{bmatrix} \frac{1}{8} \\ 0 \\ \frac{1}{10} \end{bmatrix}. \quad (8.119)$$

After $t = 0.2$, the exact solution can be seen in Figure 8.21. The figures are divided into five regions, which we denote A, B, C, D, E, from left to right.

- Regions A and E contain the initial left and right states, respectively.

Figure 8.21: Exact solution to Sod's shock tube problem at $t = 0.2$

- Region B consists of a rarefaction wave, which is a smooth transition of quantities ρ , v , and p .
- At the interface between regions C and D, we observe a contact discontinuity. This wave represents the effects of the initial separation of the left and right states travelling to the region of low pressure. Across this wave, pressure and velocity remain constant, but density is discontinuous.
- At the interface between regions D and E, a shock wave has formed due to the nonlinearities of the Euler equations. This shock wave travels with a speed s , which satisfies the Rankine-Hugoniot relationship. In this case, all considered quantities ρ , v and p present discontinuous behaviour across the wave.

8.5.3 A Riemann Solver for the Euler Equations

A more cost-effective approach is the use of approximate Riemann solvers to compute the behaviour of the left and right states of interfaces in finite volume schemes. In this section, we discuss the Riemann solver of Roe. Recall the quasilinear form of a conservation law

$$\frac{\partial \vec{u}}{\partial t} + A(\vec{u}) \frac{\partial \vec{u}}{\partial x} = 0. \quad (8.120)$$

For the Euler equations, $A(\vec{u})$ is given by Equation 8.114. The eigenvalues of A depend on the value of the solution and hence vary in time. Recall that for the Euler equations, three types of waves can be expected: contact, rarefaction and shock waves. We are interested in handling the formation of the shocks. Consider the following finite volume scheme with the forward Euler time-stepping method

$$\vec{u}_i^{t+1} = \vec{u}_i^t - \frac{\Delta t}{\Delta x} [\vec{F}_{i+1/2} - \vec{F}_{i-1/2}], \quad (8.121)$$

where $\vec{F}_{i-1/2}$ and $\vec{F}_{i+1/2}$ are interface fluxes on the left and right sides of the cell, respectively, and are defined such that

$$\vec{F}_{i-1/2} = \vec{F}(\vec{u}_{i-1/2}), \quad (8.122)$$

$$\vec{F}_{i+1/2} = \vec{F}(\vec{u}_{i+1/2}). \quad (8.123)$$

Hence, at the interfaces, we need to solve a Riemann problem of the form

$$\frac{\partial \vec{u}}{\partial t} + \frac{\partial \vec{F}}{\partial x} = 0, \quad (8.124)$$

$$\vec{u}(x, 0) = \begin{cases} \vec{u}_L & \text{if } x < 0, \\ \vec{u}_R & \text{if } x > 0. \end{cases} \quad (8.125)$$

Roe proposed considering instead a constant Jacobian matrix \tilde{A} with the following properties [22]

- \tilde{A} must have real eigenvalues and a complete set of eigenvectors such that the problem in Equation 8.120 is hyperbolic.
- \tilde{A} must be consistent with the exact problem, i.e. $\tilde{A}(\vec{u}, \vec{u}) = A(\vec{u})$.
- The resulting system must be conservative across discontinuities, i.e. $\vec{F}(\vec{u}_R) - \vec{F}(\vec{u}_L) = \tilde{A}(\vec{u}_R - \vec{u}_L)$ must be satisfied.

Roe demonstrated that the matrix can be written

$$\tilde{A} = \begin{bmatrix} 0 & 1 & 0 \\ (\gamma-1)\tilde{H} - \tilde{v}^2 - \tilde{c}^2 & (3-\gamma)\tilde{v} & \gamma-1 \\ \frac{1}{2}[(\gamma-3)\tilde{H} - \tilde{c}] & \tilde{H} - (\gamma-1)\tilde{v}^2 & \gamma\tilde{v} \end{bmatrix}, \quad (8.126)$$

which is equivalent to the original matrix we derived in Equation 8.114 when evaluated at the Roe average states given by

$$\tilde{v} = \frac{\sqrt{\rho_L}u_L + \sqrt{\rho_R}u_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}, \quad (8.127)$$

$$\tilde{H} = \frac{\sqrt{\rho_L}H_L + \sqrt{\rho_R}H_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}, \quad (8.128)$$

$$\tilde{c} = \sqrt{(\gamma-1)(\tilde{H} - \frac{1}{2}\tilde{v}^2)}. \quad (8.129)$$

The eigenvalues of \tilde{A} are

$$\tilde{\lambda}_1 = \tilde{v} - \tilde{c}, \quad \tilde{\lambda}_2 = \tilde{v}, \quad \tilde{\lambda}_3 = \tilde{v} + \tilde{c}, \quad (8.130)$$

and the matrix of eigenvectors $\tilde{Q} = [\tilde{q}_1, \tilde{q}_2, \tilde{q}_3]$ is given by

$$\tilde{Q} = \begin{bmatrix} 1 & 1 & 1 \\ \tilde{v} - \tilde{c} & \tilde{v} & \tilde{v} + \tilde{c} \\ \tilde{H} - \tilde{v}\tilde{c} & \frac{1}{2}\tilde{v}^2 & \tilde{H} + \tilde{v}\tilde{c} \end{bmatrix}. \quad (8.131)$$

The choice of a constant Jacobian matrix allows us to apply the theory in Section 8.4.1 directly. On each side of a discontinuity, the solution jump can be computed by

$$\Delta \vec{u} = (\vec{u}_R - \vec{u}_L) = \sum_{i=1}^m \tilde{\alpha}_i \tilde{q}_i, \quad (8.132)$$

where $\tilde{\alpha}_i$ is the jump in terms of the characteristic variables and \tilde{q}_i is an eigenvector of \tilde{A} . Hence, from Equation 8.132, we obtain a system of equations

$$\tilde{Q} \tilde{\vec{\alpha}} = \Delta \vec{u}, \quad (8.133)$$

where $\tilde{\vec{\alpha}} = [\tilde{\alpha}_1, \tilde{\alpha}_2, \tilde{\alpha}_3]^T$. The resulting expressions for these coefficients can be written

$$\tilde{\alpha}_1 = \frac{1}{2\tilde{c}} (\Delta u_1 \tilde{u} + \Delta u_1 \tilde{c} - \Delta u_2 - \tilde{c} \tilde{\alpha}_2), \quad (8.134)$$

$$\tilde{\alpha}_2 = \frac{\gamma-1}{\tilde{c}^2} (\Delta u_1 \tilde{H} - \Delta u_1 \tilde{u}^2 + \tilde{u} \Delta u_2 - \Delta u_3), \quad (8.135)$$

$$\tilde{\alpha}_3 = \Delta u_1 - \tilde{\alpha}_1 - \tilde{\alpha}_2. \quad (8.136)$$

Finally, the numerical flux can then be computed using

$$\vec{F}_{i\pm 1/2}(\vec{u}_L, \vec{u}_R) = \frac{1}{2} [\vec{F}(\vec{u}_L) + \vec{F}(\vec{u}_R)] - \frac{1}{2} \sum_{i=1}^3 \tilde{\alpha}_i |\tilde{\lambda}_i| \Delta u_i. \quad (8.137)$$

A disadvantage of the Roe solver is that it may violate the entropy condition in the presence of sonic rarefaction waves, where $\tilde{v} \approx \tilde{c}$ (or $\tilde{\lambda}_{1/3} \approx 0$). This occurs since the solver treats both rarefaction and shock waves as discontinuous solutions. Hence, an entropy fix needs to be added. A common approach consists of replacing the $\tilde{\lambda}_1$ or $\tilde{\lambda}_3$ eigenvalues if their absolute value is smaller than a given tolerance δ . This is known as Harten's entropy fix, which can be written [13]

$$\tilde{\lambda}_i = \begin{cases} \tilde{\lambda}_i & \text{if } |\tilde{\lambda}_i| > \delta, \\ \frac{\tilde{\lambda}_i^2}{2\delta} + 2\delta & \text{if } |\tilde{\lambda}_i| \leq \delta. \end{cases} \quad (8.138)$$

The drawback of this approach is that the value of $\delta \ll 1$ needs to be typically tuned for each specific application.



Check out the Sod's Shock-Tube Problem Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

8.6 MUSCL Schemes

We have shown in previous sections that finite-volume schemes are typically written for a general conservation law in the form

$$\frac{d\vec{u}_i}{dt} = -\frac{1}{\Delta x_i} \left[\vec{F}_{i+1/2} - \vec{F}_{i-1/2} \right], \quad (8.139)$$

where \vec{u}_i is a constant approximation of the state variable vector within the cell defined by $\Omega_i = [x_{i-1/2}, x_{i+1/2}]$, and $\vec{F}_{i+1/2}$ and $\vec{F}_{i-1/2}$ are interface flux vectors of the form

$$\vec{F}_{i+1/2} = \vec{F}(\vec{u}_{i+1/2}), \quad (8.140)$$

where $u_{i+1/2}$ is the solution at the interface between Ω_i and Ω_{i+1} defined at $x_{i+1/2}$. At this point, two values of the solution $u_{i+1/2}^L$, $u_{i+1/2}^R$ coexist, and hence the flux is obtained using a Riemann solver. In the first-order method of Godunov, the interface solution values are equal to the cell averages in the corresponding control volumes, i.e.

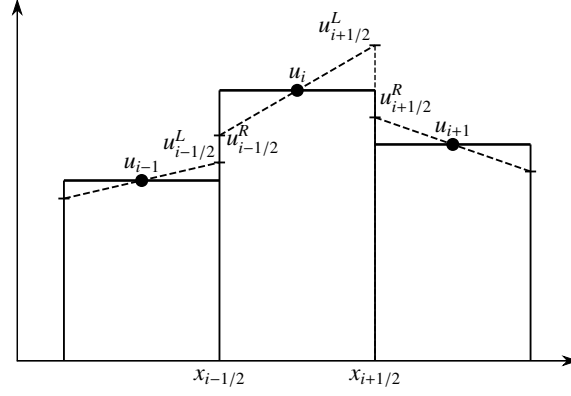
$$u_{i+1/2}^L = u_i, \quad (8.141)$$

$$u_{i+1/2}^R = u_{i+1}. \quad (8.142)$$

For a better resolution of smooth solutions, we can derive methods with orders of accuracy higher than one, allowing the numerical error to decrease significantly. This can be done by replacing the constant interface values of the solution $\vec{u}_{i+1/2}$ for a reconstructed value, which is interpolated using information from neighbouring cells. This idea, developed by van Leer [23], is an extension of Godunov's method, and is known as the MUSCL (monotone upstream-centered schemes for conservation laws) reconstruction approach. A graphical representation of the method is shown in Figure 8.22. Hence, the solution is approximated by a Taylor series expansion around the cell center x_i

$$u(x) = u_i + (x - x_i) \frac{\partial u_i}{\partial x} + \frac{1}{4} (x - x_i) \frac{\partial^2 u_i}{\partial x^2} + \dots, \quad (8.143)$$

where the derivatives are computed using cell solution differences with respect to neighbouring cells. Note that this reconstruction process can be done using additional terms in Equation 8.143 to

Figure 8.22: Linear reconstruction of the solution at interfaces $x_{i-1/2}$ and $x_{i+1/2}$

yield higher-order schemes. Hence, on the left and right sides of the interface, the solution can be found by

$$\vec{u}_{i+1/2}^L = \vec{u}_i + \frac{1}{2} \vec{\delta}_i, \quad (8.144)$$

$$\vec{u}_{i+1/2}^R = \vec{u}_{i+1} - \frac{1}{2} \vec{\delta}_{i+1}, \quad (8.145)$$

where the so-called slopes $\vec{\delta}_i$ can be defined using a blended approach with upwinding parameter b

$$\vec{\delta}_i = \frac{1}{2}(1+b)\delta\vec{u}_{i-1/2} + \frac{1}{2}(1-b)\delta\vec{u}_{i+1/2}, \quad (8.146)$$

such that $b = 1$ results in an upwind-biased approximation. Considering a linear reconstruction, the solution slopes are given by

$$\delta\vec{u}_{i+1/2} = \vec{u}_{i+1} - \vec{u}_i, \quad (8.147)$$

$$\delta\vec{u}_{i-1/2} = \vec{u}_i - \vec{u}_{i-1}. \quad (8.148)$$

Finally, the common fluxes can be computed using an appropriate Riemann solver of the form

$$\vec{F}_{i+1/2} = F(\vec{u}_{i+1/2}^L, \vec{u}_{i+1/2}^R). \quad (8.149)$$

8.6.1 Second-Order Upwind Scheme for Linear Advection

For linear advection, a common choice is the upwind Riemann flux given by

$$F^{upw}(u^L, u^R) = \alpha u^L, \quad (8.150)$$

where α is the advection velocity. At the $x_{i+1/2}$ and $x_{i-1/2}$ interfaces, the upwind flux can be found

$$F_{i+1/2}^{upw} = \alpha u_{i+1/2}^L = \alpha u_i + \frac{\alpha}{2} \delta_i, \quad (8.151)$$

$$F_{i-1/2}^{upw} = \alpha u_{i-1/2}^L = \alpha u_{i-1} + \frac{\alpha}{2} \delta_{i-1}, \quad (8.152)$$

where we choose upwind-biased slopes in Equation 8.146 with $b = 1$ such that

$$\delta_i = \delta u_{i-1/2} = u_i - u_{i-1}, \quad (8.153)$$

$$\delta_{i-1} = \delta u_{i-3/2} = u_{i-1} - u_{i-2}. \quad (8.154)$$

Hence, using Equations 8.151-8.154, the common fluxes can be written

$$F_{i+1/2} = F(u_{i+1/2}) = \alpha u_i + \frac{\alpha}{2} (u_i - u_{i-1}), \quad (8.155)$$

$$F_{i-1/2} = F(u_{i-1/2}) = \alpha u_{i-1} + \frac{\alpha}{2} (u_{i-1} - u_{i-2}), \quad (8.156)$$

which yields the following second-order upwind-biased scheme

$$\frac{du_i}{dt} = -\frac{\alpha}{2\Delta x} (3u_i - 4u_{i-1} + u_{i-2}). \quad (8.157)$$

Later in this chapter, we show results of a numerical experiment using this scheme.

8.6.2 Total Variation Diminishing

The total variation (TV) of a scalar conservation law such as the linear advection equation is given by [7]

$$TV(u) = \int \left| \frac{\partial u}{\partial x} \right| dx, \quad (8.158)$$

which can be written for the discrete approximation

$$TV(u) = \sum_{i=1}^N |u_{i+1} - u_i|, \quad (8.159)$$

where N is the number of finite volume cells. A numerical method is said to be total variation diminishing (TVD) if the solution does not spontaneously concentrate, i.e.

$$TV(u^t + 1) \leq TV(u^t). \quad (8.160)$$

Harten [6] proved that TVD schemes are monotonicity-preserving, meaning that in time, maxima does not increase, minima does not decrease, and no new extrema are generated in the solution. It turns out that this property can only be attributed to certain types of schemes. Godunov's theorem states [5]

Theorem 8.6.1 Linear numerical schemes for solving partial differential equations (PDE's), having the property of not generating new extrema (monotone scheme), can be at most first-order accurate.

Hence, while higher-order schemes are more accurate for smooth solutions, they introduce spurious oscillations in the presence of discontinuities, shocks and large solution gradients and are thus not TVD. Hence, TVD schemes are at most first-order. Different approaches can be used to ensure the TVD property for high-order numerical methods under these circumstances. In the next section, we discuss one of these techniques, known as flux/slope limiters.

8.6.3 Limiters

Introducing limiters to high-order formulations prevents the generation of oscillations in regions where nonsmooth solutions are present. Recall that Godunov's theorem states that only first-order accurate schemes are able to produce solutions without generating new extrema. Hence, limiters are used as a switch mechanism which transforms the high-order numerical scheme into a first-order method in the presence of discontinuities. This is done by evaluating and comparing slopes between neighbouring cells. In this section, we introduce limiters to the MUSCL reconstruction framework in Section 8.6.

Table 8.1: Common forms of limiter functions $\phi(r)$

Name	$\phi(r)$
minmod	$\max[0, \min(1, r)]$
van Leer	$\frac{r+ r }{1+ r }$
superbee	$\max[0, \min(2r, 1), \min(r, 2)]$

The MUSCL formulation in Equation 8.146 can be rewritten for a scalar conservation law using limited slopes

$$\bar{\delta}_i = \frac{1}{2}(1+b)\phi_{i-1/2}^+ \delta u_{i-1/2} + \frac{1}{2}(1-b)\phi_{i+1/2}^- \delta u_{i+1/2}, \quad (8.161)$$

where the limiting functions can be defined using ratios of the slopes with neighbouring cells. At interface x_{i+m} , the slope limiter is given by

$$\phi_{i+m}^\pm = \phi(r_{i+m}^\pm), \quad r_{i+m}^\pm = \frac{\delta u_{i+m\pm 1}}{\delta u_{i+m}}. \quad (8.162)$$

Limiters are designed to treat similarly both upwind and downwind slopes by satisfying the symmetry condition

$$\frac{\phi(r)}{r} = \phi\left(\frac{1}{r}\right), \quad (8.163)$$

which allows us to simplify Equation 8.161 to

$$\bar{\delta}_i = \frac{\phi(r_{i-1/2}^+)}{2} \left[(1+b)\delta u_{i-1/2} + \frac{1}{r_{i-1/2}^+} \delta u_{i+1/2} \right], \quad (8.164)$$

where

$$r_{i-1/2}^+ = \frac{\delta u_{i+1/2}}{\delta u_{i-1/2}} = \frac{u_{i+1} - u_i}{u_i - u_{i-1}}. \quad (8.165)$$

For the second-order upwind-biased scheme in Equation 8.157, we can obtain a scheme with limiting functions using interface fluxes of the form

$$F_{i+1/2} = F(u_{i+1/2}) = \alpha u_i + \frac{\alpha}{2} \phi(r_{i-1/2}^+) (u_i - u_{i-1}), \quad (8.166)$$

$$F_{i-1/2} = F(u_{i-1/2}) = \alpha u_{i-1} + \frac{\alpha}{2} \phi(r_{i-3/2}^+) (u_{i-1} - u_{i-2}). \quad (8.167)$$

Clearly, in the case $\phi(r) = 0$, the flux reduces to the first-order upwind scheme. This occurs particularly when the limiter detects a change in the slope (negative r). Some slope limiting functions are shown in Table 8.1. In the next section, we show applications of these limiters in the context of linear advection and the Euler equations.

8.6.4 Numerical Examples

Linear Advection

Consider the following advection problem

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0, \quad (8.168)$$

on a grid with $x \in [0, 2]$ with periodic boundary conditions. The initial conditions at $t = 0$ are given by

$$u(x, 0) = \begin{cases} e^{-20(x-0.5)^2} & \text{if } x < 1.2, \\ 1 & \text{if } 1.2 < x < 1.5, \\ 0 & \text{otherwise.} \end{cases} \quad (8.169)$$

which consist of a smooth gaussian profile and a step function. Due to the nature of the equation, we expect a translation of the initial condition through the domain without any deformation. At $t = 2$, a complete cycle has occurred and it is expected that $u_i^{t=2} \approx u(x_i, 0)$. Using the second-order advection scheme with $b = 1$ and $N = 300$ cells, the solution is shown in Figure 8.23. In the smooth region of the domain, the second-order scheme does a good job representing the solution, but we observe that the discontinuities on the right side of the domain contain large oscillations that overshoot and undershoot the exact solution.

Implementation of the function in Table 8.1 shows the monotonicity of the slope-limited methods. Clearly, the resolution of the shock does not include oscillations, and some limiters introduce more dissipation than others. In all limited cases, observe the crest of the Gauss wave. Due to the slope change of the solution in that region, we have therein introduced additional error.

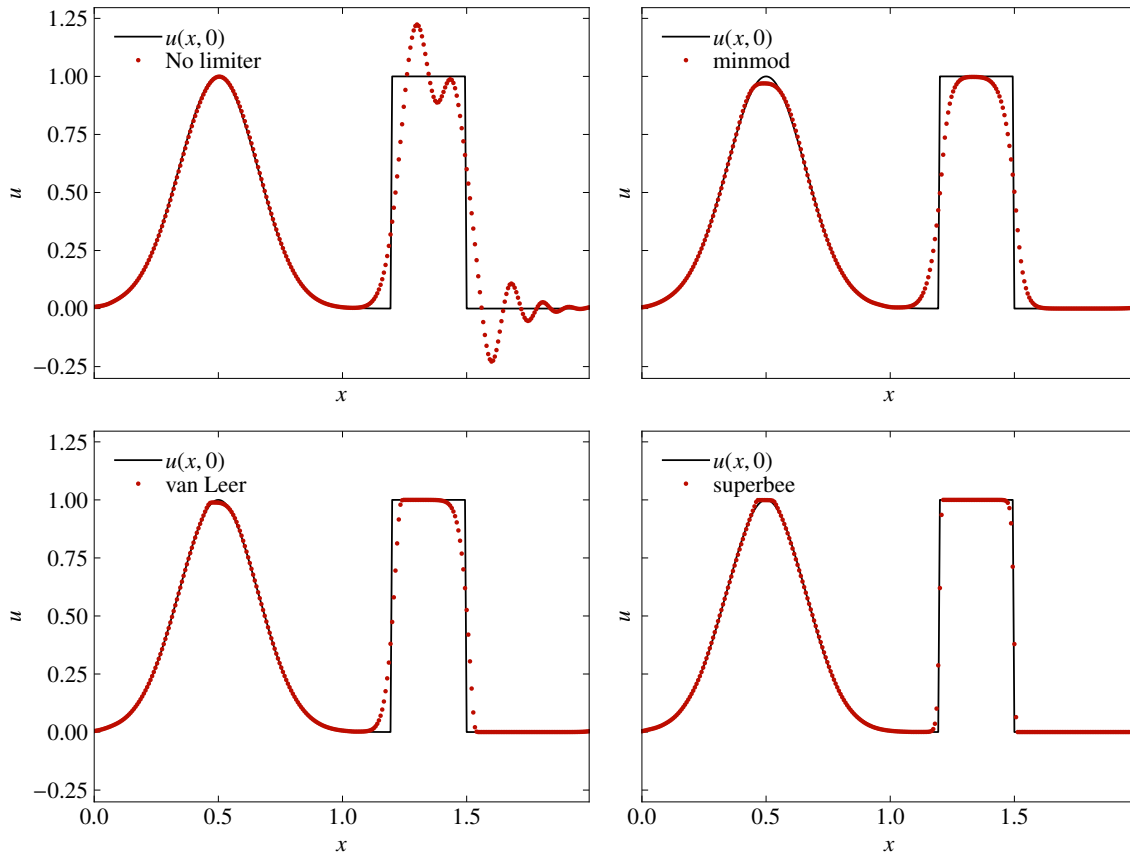


Figure 8.23: Second-order upwind-biased advection scheme (upper left) using minmod (upper right), van Leer (lower left) and superbee (lower right) limiters

A comparison of the TVD properties of the scheme and other additional assessments can be made using the MUSCL Schemes Jupyter notebook.



Check out the MUSCL Schemes Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

Sod's Shock-Tube Problem

Consider Sod's shock-tube problem with initial conditions

$$\vec{u}_L = \begin{bmatrix} \rho_L \\ v_L \\ p_L \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad \text{and} \quad \vec{u}_R = \begin{bmatrix} \rho_R \\ v_R \\ p_R \end{bmatrix} = \begin{bmatrix} \frac{1}{8} \\ 0 \\ \frac{1}{10} \end{bmatrix}. \quad (8.170)$$

on a domain $x \in [0, 1]$ and $t \in (0, 0.2]$. Figure 8.24 shows a comparison between Godunov's first-order scheme and a linear MUSCL-reconstructed method with different limiters. We note that this problem is unstable for the MUSCL scheme if no limiter function is added. The analysis of these results is similar to the advection example and is left as an exercise to the student. The associated Jupyter notebook shows the implementation of these functions for the Euler equations.



Check out the Sod's Shock-Tube Problem Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

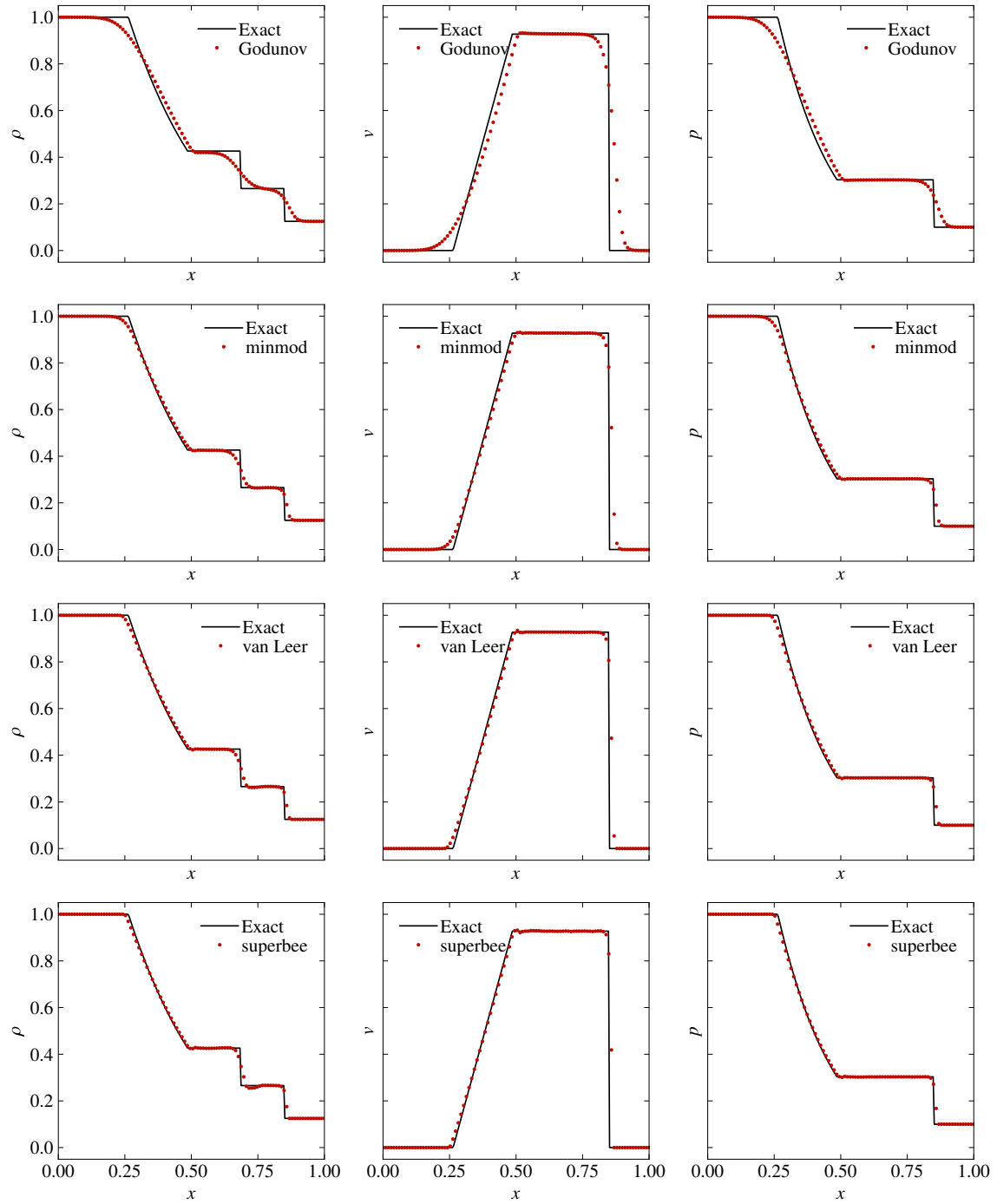


Figure 8.24: Results of Sod's shock tube problem at $t = 0.2$ with initial conditions in Equation 8.170

9. Consistency, Stability, Convergence

Now that we have derived a few different schemes for solving linear advection, Burgers equation, and linear diffusion, we should now ask ourselves whether they will give us accurate predictions, and are there any restrictions on when they can be used. We have already discussed the importance of the order of accuracy, which governs the rate at which the error will converge to zero. Now we will introduce the concepts of consistency, stability, and convergence. If we can prove that our numerical scheme satisfies all three of these properties, we can be confident that it is a promising approach for the Euler or Navier-Stokes equations. In the current section, we will explore these properties in the context of finite difference methods, but the exact same steps can be taken to check whether a finite volume method is suitable or not. Some additional references on these topics include [1, 7].

9.1 Consistency

A numerical scheme is consistent if it recovers the exact initial partial differential equation as the grid spacing and time step size are reduced. In other words, the truncation error of the scheme must go to zero in the limit $\Delta x \rightarrow 0$ and $\Delta t \rightarrow 0$. This is usually the case, and it is left as an exercise for the reader to check that our previous finite difference schemes for linear advection, Burgers equation, and linear diffusion satisfy this property.

However, this is not always the case. To demonstrate this, we consider the Dufort-Frankel scheme for linear diffusion

$$u_i^{t+1} = u_i^{t-1} + \frac{2\beta\Delta t}{\Delta x^2} (u_{i-1}^t - u_i^{t+1} - u_i^{t-1} + u_{i+1}^t) + \mathcal{O}(\Delta x^2, \Delta t^2, (\Delta t/\Delta x)^2). \quad (9.1)$$

This differs from our example scheme for linear diffusion in that it uses central differences for the time-derivative, and when computing the second-derivative in space, it uses the solution at the current grid point at the previous and current time steps. While this scheme may have some useful properties, we note that it has a peculiar term in the truncation error of $\mathcal{O}((\Delta t/\Delta x)^2)$. This can be obtained from a Taylor series expansion of the second derivative. This is concerning, as for any scheme to be consistent with the original partial differential equation we require the truncation error

to go to zero. However, if we use a naive approach and simply refine Δx and Δt at the same rate, this term will not go to zero, and the scheme will not be consistent. For example, if we reduced both the grid spacing and time step size by a half, this $\mathcal{O}((\Delta t/\Delta x)^2)$ will remain the same. Hence, in order for the Dufort-Frankel scheme to be consistent with our original partial differential equation, we should refine the grid spacing *faster* than the time step size. For example, if we reduce Δt by a half we should reduce Δx by a factor of four. This does not mean the Dufort-Frankel scheme is bad, per-se, but it does mean that care needs to be taken when using it.

9.2 Stability

If we can demonstrate our schemes are consistent, then we know that in the limit $\Delta x \rightarrow 0$ and $\Delta t \rightarrow 0$ we recover the exact partial differential equation. However, this is impossible to achieve in practice as it would require an infinite number of grid points and time steps. When considering stability, we are concerned with whether our numerical scheme will provide *physical* solutions when both Δx and Δt are finite. Let's start by considering what we mean by a physical solution.

In order to advance our solution in time, we start with some initial condition. Then, by inserting this initial condition into our scheme, we approximate the solution at the next timestep $t + \Delta t$. Then, we insert this approximation back into our scheme to approximate the solution at time $t + 2\Delta t$, and this process is repeated over and over again until we reach our final desired time. Hence, the way we advance our simulation in time is effectively a feedback loop, with the output of each time step being recycled back through the numerical scheme to get the solution at each consecutive time step.

As an analogy, we can consider what happens in other simple feedback loops, such as a microphone and speaker. When a performer sings into a microphone, their voice is amplified and played back through the speaker. We expect that this produces a physical replication of their voice, just at a louder volume for the audience. However, if the sound from the speaker is louder than the singer's voice at the microphone, it will get amplified, played through the speaker at a louder volume, and this cycle then repeats. This results in feedback noise, usually a high-pitched ringing that sounds nothing like the original performer, and usually happens when the performer moves too close to the speaker.

Since our numerical scheme is applied as a feedback loop, the exact same kind of thing can happen. If it amplifies our solution each time step, then the solution will continue to grow, eventually leading to non-physical values such as near-infinite density or pressure. This is colloquially referred to as the solution *blowing up*. In contrast, if the scheme damps our solution at each time step, it will tend towards physical values, such as the background density or pressure. While this is perhaps not desirable in terms of accuracy, which will be explored later, it is a desirable property in that the solution remains stable and bounded between the initial condition and background state.

Proving stability for non-linear problems, such as Burgers equation, is a relatively daunting task. However, if we restrict ourselves to linear equations, such as linear advection or linear diffusion, then stability can be explored more readily. In order to do this, we introduce *Von Neumann Analysis*, also commonly referred to as Fourier Analysis. We first assume that our solution $u(x, t)$ can be represented via a Fourier Series, such that

$$u(x, t) = \sum_m b_m(t) e^{i\kappa_m x}, \quad (9.2)$$

where $b_m(t)$ is the Fourier coefficients that vary with time as the solution evolves, κ_m is a wavenumber, and in this context $i = \sqrt{-1}$. This is the exact same as a conventional Fourier series, with the exception that the coefficients are a function of time describing the time evolution of the system of equations. Furthermore, we take

$$\kappa_m = \frac{2\pi m}{2L}, \quad m = 0, 1, 2, \dots, M, \quad (9.3)$$

where M is chosen based on the maximum wavenumber that can be represented on the grid based on its Nyquist criteria, and L is the length of the domain. Hence, small values of κ_m correspond to large waves, and large values of κ_m correspond to very compact waves.

Following this approach, we are taking our solution and decomposing it into a number of different waves via a Fourier series. Now, since we have restricted ourselves to linear systems of equations, we can apply the property of superposition. This means that we can analyze each wave independently as a function of time, and the final solution is simply the superposition of all of these waves. This allows us to analyze the behaviour of our numerical scheme for each wavenumber independently, since they are not coupled. Hence, we can write the solution for one particular wave of the Fourier series as

$$u_m(x, t) = b_m(t) e^{i\kappa_m x}, \quad (9.4)$$

where the complete solution is

$$u(x, t) = \sum_m u_m(x, t). \quad (9.5)$$

We will also assume that the time-dependence of the solution is also wavelike with a prescribed frequency in time such that

$$u_m(x, t) = e^{at} e^{i\kappa_m x}, \quad (9.6)$$

where a is a complex number, referred to as the numerical frequency, that describes how the solution evolves in time. In order to justify this assumption we can consider the linear advection equation applied to an arbitrary wavenumber κ_m . We note that this wave will propagate at velocity α from left to right. Now, if we consider some fixed point in space, denoted by $u_1 t$, we note that the value of the solution in time will also behave like a sine wave. Hence, at a fixed point in time, the solution has a wavelike structure in space and, at a fixed point in space, the solution has a wavelike structure in time. Hence, the dual wavelike structure taken for $u_m(x, t)$.

With the wavelike structure of the solution described, we can now explore how that wave will change with time. Of primary importance, at least in terms of stability, is to determine whether the wave will be amplified or damped as the solution evolves. Our objective in this section is to determine whether, and under what conditions, numerical schemes satisfy this stability condition. We note that the solution at time $t + \Delta t$ is simply

$$u_m(x, t + \Delta t) = e^{a(t+\Delta t)} e^{i\kappa_m x}, \quad (9.7)$$

which can be re-written as

$$u_m(x, t + \Delta t) = e^{at} e^{a\Delta t} e^{i\kappa_m x}, \quad (9.8)$$

and, in order for the magnitude of the solution to not be amplified at the next time step, we have the stability constraint

$$|e^{a\Delta t}| \leq 1, \quad (9.9)$$

which describes a unit circle in the complex plane. Referred to as the amplification factor, we will next determine under what conditions our numerical schemes satisfy this stability constraint.

9.2.1 Explicit Linear Advection

For a methodological approach, we will break von Neumann down into a number of steps.

Step 1: Choose the Discrete Scheme

The first step in von Neumann analysis is to determine what scheme we are interested in analyzing. In this case, we will consider our simple first-order scheme for linear advection

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} + \alpha \frac{u_i^t - u_{i-1}^t}{\Delta x} = 0. \quad (9.10)$$

Step 2: Apply the Wavelike Solution

Since we know the prescribed wave-like form of the solution, the grid spacing Δx , and the time step size Δt , we can write expressions for the solution for a particular wavenumber κ_m at each grid point and time level as

$$u_i^t = e^{at} e^{i\kappa_m x}, \quad (9.11)$$

$$u_i^{t+1} = e^{a(t+\Delta t)} e^{i\kappa_m x}, \quad (9.12)$$

$$u_{i-1}^t = e^{at} e^{i\kappa_m (x-\Delta x)}. \quad (9.13)$$

Substituting these into our finite difference approximation yields

$$\frac{e^{a(t+\Delta t)} e^{i\kappa_m x} - e^{at} e^{i\kappa_m x}}{\Delta t} + \alpha \frac{e^{at} e^{i\kappa_m x} - e^{at} e^{i\kappa_m (x-\Delta x)}}{\Delta x} = 0. \quad (9.14)$$

Step 3: Solve for the Amplification Factor

Noting that all of these terms has a common factor of $e^{at} e^{i\kappa_m x}$ we can simply divide through yielding

$$\frac{e^{a\Delta t} - 1}{\Delta t} + \alpha \frac{1 - e^{-i\kappa_m \Delta x}}{\Delta x} = 0. \quad (9.15)$$

Rearranging yields

$$e^{a\Delta t} = 1 - \sigma + \sigma e^{-i\kappa_m \Delta x}, \quad (9.16)$$

where

$$\sigma = \frac{\alpha \Delta t}{\Delta x}, \quad (9.17)$$

is the *Courant-Friedrichs-Lewy* (CFL) number. Hence, the amplification factor of our first-order linear advection scheme is

$$|e^{a\Delta t}| = |1 - \sigma + \sigma e^{-i\kappa_m \Delta x}|, \quad (9.18)$$

and our scheme will be stable whenever this is contained within the unit circle in the complex plane. We note that this is a function of two parameters, specifically the wavenumber and the CFL number. Hence, we expect that the amount our solution gets amplified/damped each time step will depend on these two parameters.

Step 4: Determine the Stability Conditions

Based on these results, we can conclude that the first-order finite difference scheme for linear advection is stable whenever

$$0 \leq \sigma \leq 1. \quad (9.19)$$

That is, it is only stable for CFL numbers less than one. Hence, for a given grid spacing and advection velocity, there is a limit on how large the time step can be. This clearly has implications in terms of computational cost, as the smaller the time step is, the more steps must be taken to reach a desired final solution time. Schemes of this type are referred to as being *conditionally stable*.

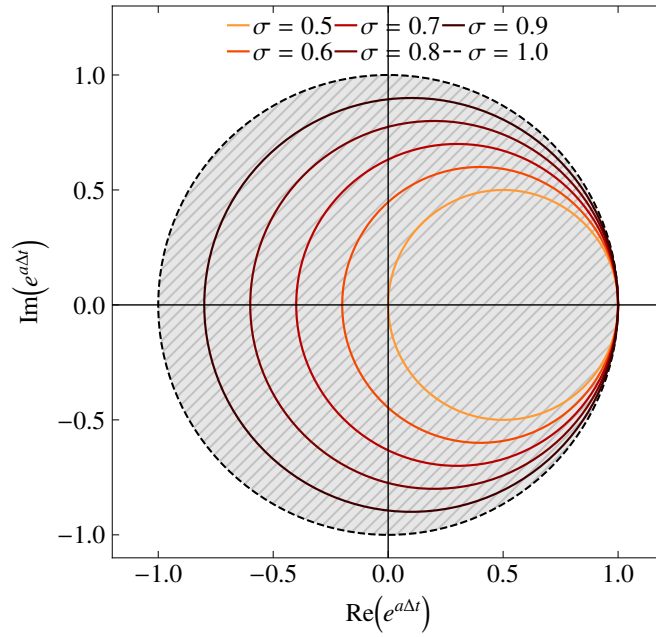


Figure 9.1: Stability region for the explicit linear advection scheme.

9.2.2 Implicit Linear Advection

In the last section, we saw that our first-order finite difference scheme has a stability limit. In this section, we will explore a slightly different first-order scheme for linear advection.

Step 1: Choose the Discrete Scheme

In this case, we use the following finite difference approximation of the linear advection equation

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} + \alpha \frac{u_i^{t+1} - u_{i-1}^{t+1}}{\Delta x} = 0, \quad (9.20)$$

noting that it is simply the original scheme, but we evaluate the spatial derivative at the next time step rather than the current time step.

Step 2: Apply the Wavelike Solution

Our expressions for wavelike solutions at each point are

$$u_i^t = e^{at} e^{i\kappa_m x}, \quad (9.21)$$

$$u_i^{t+1} = e^{a(t+\Delta t)} e^{i\kappa_m x}, \quad (9.22)$$

$$u_{i-1}^{t+1} = e^{a(t+\Delta t)} e^{i\kappa_m(x-\Delta x)}. \quad (9.23)$$

Substituting these into our discrete scheme yields

$$\frac{e^{a(t+\Delta t)} e^{i\kappa_m x} - e^{at} e^{i\kappa_m x}}{\Delta t} + \alpha \frac{e^{a(t+\Delta t)} e^{i\kappa_m x} - e^{a(t+\Delta t)} e^{i\kappa_m(x-\Delta x)}}{\Delta x} = 0. \quad (9.24)$$

Step 3: Solve for the Amplification Factor

Again, we note a common factor of $e^{at} e^{i\kappa_m x}$ in all terms, allowing us to divide through yielding

$$\frac{e^{a\Delta t} - 1}{\Delta t} + \alpha \frac{e^{a\Delta t} - e^{a\Delta t} e^{-i\kappa_m \Delta x}}{\Delta x} = 0. \quad (9.25)$$

Rearranging yields

$$e^{a\Delta t} = \frac{1}{1 + \sigma (1 - e^{-i\kappa_m \Delta x})}, \quad (9.26)$$

where σ is again the CFL number. Hence, the amplification factor is

$$|e^{a\Delta t}| = \left| \frac{1}{1 + \sigma (1 - e^{-i\kappa_m \Delta x})} \right|. \quad (9.27)$$

Step 4: Determine the Stability Conditions

Based on these results, we can conclude that this finite difference scheme for the linear advection equation is stable provided

$$0 \leq \sigma \leq \infty. \quad (9.28)$$

Hence, we are able to take arbitrarily large time steps and maintain stability using this approach. Schemes of this type are referred to as being *unconditionally stable*.

R Although this scheme is unconditionally stable, making it appealing since it allows for arbitrarily large time-steps, it also becomes more difficult/expensive for each time-step. This will be explored in the forthcoming time stepping section.

9.2.3 Explicit Linear Diffusion

Similar to the linear advection equation, we can also use von Neumann analysis to analyze the stability of the linear diffusion equation.

Step 1: Choose the Discrete Scheme

We will start with the example scheme we derived in the finite difference section

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} - \beta \frac{u_{i-1}^t - 2u_i^t + u_{i+1}^t}{\Delta x^2} = 0. \quad (9.29)$$

Step 2: Apply the Wavelike Solution

Our expressions for wavelike solutions at each point are

$$u_i^t = e^{at} e^{i\kappa_m x}, \quad (9.30)$$

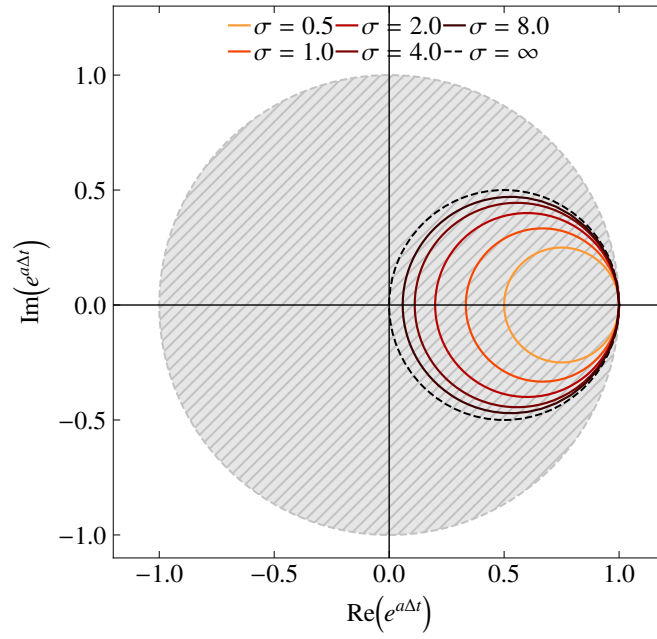


Figure 9.2: Stability region for the implicit linear advection scheme.

$$u_i^{t+1} = e^{a(t+\Delta t)} e^{i\kappa_m x}, \quad (9.31)$$

$$u_{i-1}^t = e^{at} e^{i\kappa_m (x-\Delta x)}, \quad (9.32)$$

$$u_{i+1}^t = e^{at} e^{i\kappa_m (x+\Delta x)}. \quad (9.33)$$

Substituting these into our discrete scheme yields

$$\frac{e^{a(t+\Delta t)} e^{i\kappa_m x} - e^{at} e^{i\kappa_m x}}{\Delta t} - \beta \frac{e^{at} e^{i\kappa_m (x-\Delta x)} - 2e^{at} e^{i\kappa_m x} + e^{at} e^{i\kappa_m (x+\Delta x)}}{\Delta x^2} = 0. \quad (9.34)$$

Step 3: Solve for the Amplification Factor

Again we note a common factor of $e^{at} e^{i\kappa_m x}$ in all terms, allowing us to divide through yielding

$$\frac{e^{a\Delta t} - 1}{\Delta t} - \beta \frac{e^{-i\kappa_m \Delta x} - 2 + e^{i\kappa_m \Delta x}}{\Delta x^2} = 0. \quad (9.35)$$

Rearranging yields

$$e^{a\Delta t} = 1 + r (e^{-i\kappa_m \Delta x} - 2 + e^{i\kappa_m \Delta x}), \quad (9.36)$$

where

$$r = \frac{\beta \Delta t}{\Delta x^2}, \quad (9.37)$$

is similar to the CFL number but for diffusion rather than advection. Finally, the amplification factor is

$$|e^{a\Delta t}| = |1 + r(e^{-i\kappa_m\Delta x} - 2 + e^{i\kappa_m\Delta x})|. \quad (9.38)$$

Step 4: Determine the Stability Conditions

Based on this amplification factor, we demonstrate graphically that this scheme will be stable provided

$$0 \leq r \leq \frac{1}{2}. \quad (9.39)$$

We note that this scheme is *conditionally stable*, similar to the first linear advection scheme we considered. This means if the grid is refined then the time step size must be reduced accordingly to maintain stability. However, we note that the factor of Δx^2 in r will require the time step size to be reduced with the square of the grid spacing, which can become expensive on finer meshes.

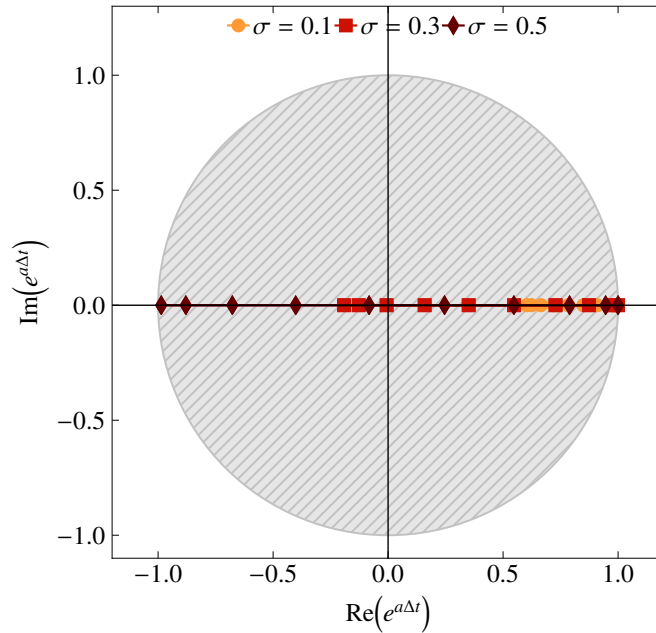


Figure 9.3: Stability region for the explicit linear diffusion scheme.



The Navier-Stokes equations have both advective and diffusive terms. Hence, the time step is usually limited by the stricter of the stability constraints of the advective and diffusive schemes that are being used.

9.2.4 Implicit Linear Diffusion

Similar to the modified linear advection scheme, we can also modify our initial linear diffusion scheme by evaluating the spatial operator at the future unknown solution time.

Step 1: Choose the Discrete Scheme

This yields the following scheme

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} - \beta \frac{u_{i-1}^{t+1} - 2u_i^{t+1} + u_{i+1}^{t+1}}{\Delta x^2} = 0. \quad (9.40)$$

Step 2: Apply the Wavelike Solution

Our expressions for wavelike solutions at each point are

$$u_i^t = e^{at} e^{i\kappa_m x}, \quad (9.41)$$

$$u_i^{t+1} = e^{a(t+\Delta t)} e^{i\kappa_m x}, \quad (9.42)$$

$$u_{i-1}^{t+1} = e^{a(t+\Delta t)} e^{i\kappa_m (x-\Delta x)}, \quad (9.43)$$

$$u_{i+1}^{t+1} = e^{a(t+\Delta t)} e^{i\kappa_m (x+\Delta x)}. \quad (9.44)$$

Substituting these into our discrete scheme yields

$$\frac{e^{a(t+\Delta t)} e^{i\kappa_m x} - e^{at} e^{i\kappa_m x}}{\Delta t} - \beta \frac{e^{a(t+\Delta t)} e^{i\kappa_m (x-\Delta x)} - 2e^{a(t+\Delta t)} e^{i\kappa_m x} + e^{a(t+\Delta t)} e^{i\kappa_m (x+\Delta x)}}{\Delta x^2} = 0. \quad (9.45)$$

Step 3: Solve for the Amplification Factor

Again we note a common factor of $e^{at} e^{i\kappa_m x}$ in all terms, allowing us to divide through yielding

$$\frac{e^{a\Delta t} - 1}{\Delta t} - \beta \frac{e^{a\Delta t} e^{-i\kappa_m \Delta x} - 2e^{a\Delta t} + e^{a\Delta t} e^{i\kappa_m \Delta x}}{\Delta x^2} = 0. \quad (9.46)$$

Rearranging yields

$$e^{a\Delta t} = \frac{1}{[1 - r(e^{-i\kappa_m \Delta x} - 2 + e^{i\kappa_m \Delta x})]}, \quad (9.47)$$

where r is the same as the original linear diffusion scheme. Finally, the amplification factor is

$$|e^{a\Delta t}| = \left| \frac{1}{[1 - r(e^{-i\kappa_m \Delta x} - 2 + e^{i\kappa_m \Delta x})]} \right|. \quad (9.48)$$

Step 4: Determine the Stability Conditions

Based on the form of the amplification factor, we can conclude that this scheme will be stable for

$$0 \leq r \leq \infty. \quad (9.49)$$

Hence, this linear diffusion scheme is *unconditionally stable*.

9.3 Convergence

If a scheme is consistent, recovering the exact partial differential equation in the limit $\Delta x \rightarrow 0$ and $\Delta t \rightarrow 0$, and stable, in that the approximate solution does not grow unbounded with time, then Lax's equivalence theorem can be applied. In summary, the theorem states that when given a properly-posed initial value problem, and a numerical scheme that is consistent, stability is the necessary and sufficient condition for convergence. In other words, if we can show that our numerical schemes are consistent and stable, then we can be sure that it will *converge* to the true solution of the partial differential equation in the limit $\Delta x \rightarrow 0$ and $\Delta t \rightarrow 0$.

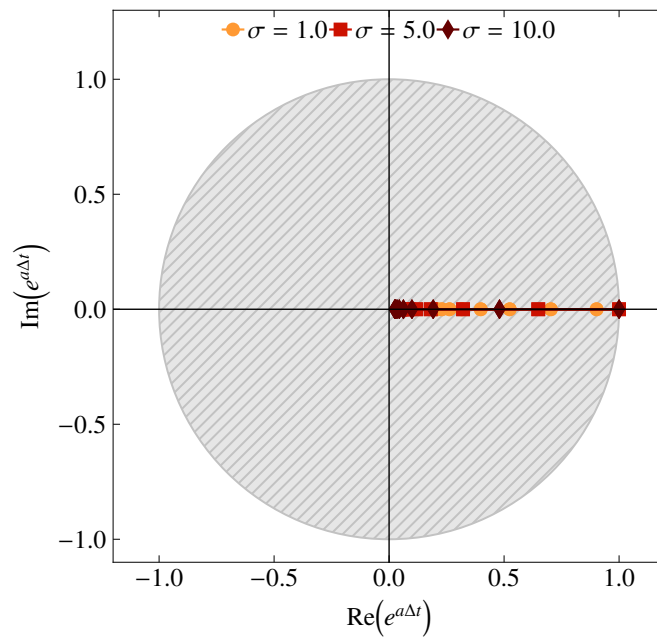


Figure 9.4: Stability region for the implicit linear diffusion scheme.

10. Spectral Properties

In CFD, the numerical error introduced by a scheme is typically classified into two general types, dissipation error and dispersion error. In this section, we will discuss how to quantify these two types of error, and how they effect different types of solutions.

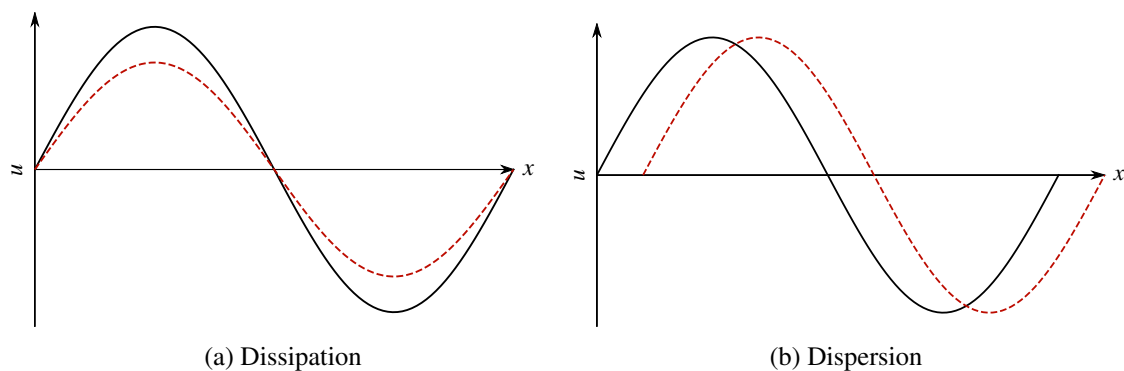


Figure 10.1: Types of numerical error

10.1 Dissipation Error

We have already discussed dissipation error in the context of stability, stating that in order for a scheme to be stable, it must be dissipative or neutrally dissipative. That is, the amplitude of the solution must remain the same or be reduced after each time step. We have shown this via von Neumann analysis to be the amplification factor $|e^{a\Delta t}|$, which was a function of the scheme we are using, and the wavenumber in space κ_m . While a scheme being dissipative is inherently linked with stability, it also introduces number error. For example, for the linear advection equation, we know that the exact solution should have no dissipation, it should be simply the advection of the initial condition with the prescribed advection velocity. However, with our initial finite difference scheme, we found that this was not the case, and the amplitude of the waves is decreased as time goes on.

It turns out that all the information we need to understand the dissipation error of the linear advection equation is encoded in the amplification factors we found in the von Neumann analysis section, $|e^{a\Delta t}|$. For example, for the finite difference scheme

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} + \alpha \frac{u_i^t - u_{i-1}^t}{\Delta x} = 0. \quad (10.1)$$

we found that the amplification factor was

$$|e^{a\Delta t}| = |1 - \sigma + \sigma e^{-i\kappa_m \Delta x}|. \quad (10.2)$$

Recall that, for a given wavenumber κ_m and CFL number σ , this told us how much the solution will be damped over a time step. Hence, we can plot this for a range of permissible wavenumbers in the range $\kappa_m \Delta x \in [0, \pi]$ and CFL numbers within the stability limit $\sigma \in [0, 1]$. A plot of this is shown in Figure 10.2. We can observe a few interesting features that define the dissipation error of this particular scheme. First, we note that when the wave size is large relative to the grid spacing, the wave is well resolved regardless of the CFL number. However, as we move to larger wavenumbers, there is significant numerical dissipation. Also, the amount of numerical dissipation in this region depends on the CFL number, with larger CFL numbers corresponding to less dissipation. Finally, when the CFL number approaches unity, the numerical dissipation approaches zero for all wavenumbers. Hence, when this CFL number is used, we recover the exact dissipation relation for linear advection, that the waves do not get damped. The dissipation error of this scheme becomes particularly important in the context of turbulent flows. In this case, large scale structures in the turbulent flow have small wavenumbers relative to the grid spacing. This means that our large scale features will be simulated with relative accuracy. However, small scale turbulent structures that are of a size proportional to the grid spacing have high relative wavenumbers. This means that these fine-scale structures will be heavily dissipated by this first-order scheme. Let's look at a

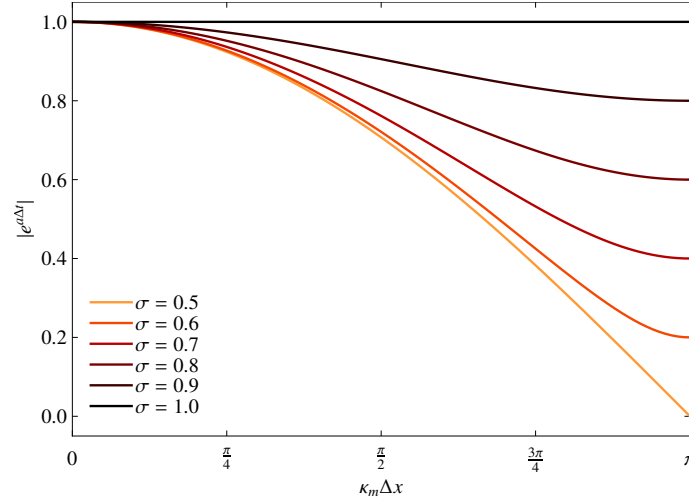


Figure 10.2: Amplification factor vs wavenumber for the explicit advection scheme

second example, our implicit finite difference scheme

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} + \alpha \frac{u_i^{t+1} - u_{i-1}^{t+1}}{\Delta x} = 0, \quad (10.3)$$

that had the amplification factor

$$|e^{a\Delta t}| = \left| \frac{1}{1 + \sigma(1 - e^{-i\kappa_m \Delta x})} \right|. \quad (10.4)$$

We can generate the exact same type of plot for permissible wave numbers $\kappa_m \Delta x \in [0, \pi]$ and, since this scheme is unconditionally stable, we will look at a few CFL numbers in the range $\sigma \in [0, 10]$, as shown in Figure 10.4. We note for this scheme, similar to the explicit scheme, that when the wave size is large relative to the grid spacing, it is well resolved regardless of the CFL number. However, there is again a strong dependence of the numerical dissipation on the CFL number for high wavenumbers. As the wavenumber gets larger or as the CFL number gets larger, the amount of numerical dissipation increases rapidly. Hence, while this scheme was found to be unconditionally stable, using large CFL numbers introduces significant numerical dissipation to all but the largest structures in the flow. Hence, this scheme is typically only used for solving steady-state problems that are not a function of time.

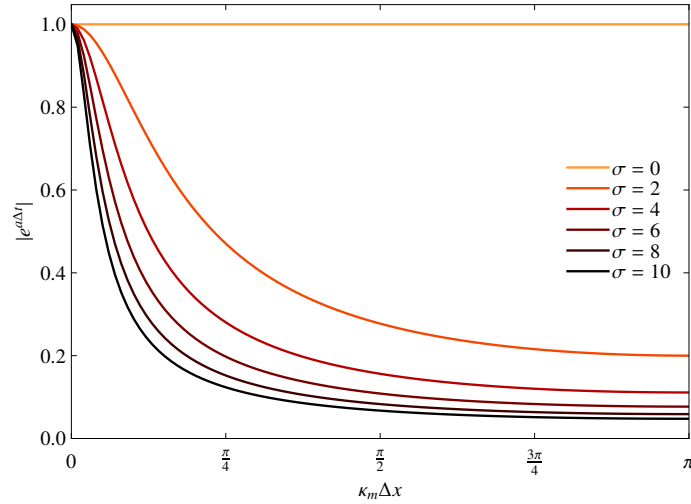


Figure 10.3: Amplification factor vs wavenumber for the implicit advection scheme



Check out the Von Neumann Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

10.2 Dispersion Error

From the linear advection equation, we know that the solution should translate at an exact wave speed α . However, our numerical schemes will often introduce another error whereby the numerical wave does not move at the correct velocity. As a result, we often observe that the numerical solution is out of phase with the exact solution, which is referred to as dispersion error. Similar to the dissipation error, dispersion error can also be found from our results from von Neumann analysis. Recall that our assumed form of the solution for any isolated wave component of the full solution was

$$u_m(x, t) = e^{at} e^{i\kappa_m x}, \quad (10.5)$$

and recall the expression for the linear advection equation was

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0. \quad (10.6)$$

If we substitute our wavelike solution we get

$$ae^{at} e^{i\kappa_m x} + i\alpha \kappa_m e^{at} e^{i\kappa_m x} = 0. \quad (10.7)$$

We note that this will be solved exactly when

$$a = -i\alpha\kappa_m. \quad (10.8)$$

which is called the exact dispersion relation. This tells us that the exact frequency in time, which is described by a , is proportional to the wave speed and the wavenumber. Intuitively this makes sense as the faster a wave moves, the more the solution at a point should move up and down, and the higher the wavenumber, the more quickly the wave will move up and down in time as well. However, our numerical scheme will usually not return a value of a that matches this dispersion relation exactly. Hence, for a given wavenumber, we will get back a frequency that has some error, implying that the wave is moving at the wrong speed, which is referred to as dispersion error. Over a finite amount of time Δt , based on the exact dispersion relation, we would expect the wave to move a distance

$$a\Delta t = -i\alpha\kappa_m\Delta t. \quad (10.9)$$

We note that we can also get to our numerical frequency from von Neumann analysis. For example, for the finite difference scheme

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} + \alpha \frac{u_i^t - u_{i-1}^t}{\Delta x} = 0, \quad (10.10)$$

we found

$$e^{a\Delta t} = 1 - \sigma + \sigma e^{-i\kappa_m\Delta x}, \quad (10.11)$$

which was the last step before getting the amplification factor. We can start by splitting the exponential term up into its real and imaginary parts

$$e^{\Re(a)\Delta t} e^{\Im(a)\Delta t} = 1 - \sigma + \sigma e^{-i\kappa_m\Delta x}. \quad (10.12)$$

We first note that the real part of this $e^{\Re(a)\Delta t}$ is responsible for the amplification/dissipation of the solution, and is what is extracted in the dissipation section. In contrast, the imaginary part $e^{\Im(a)\Delta t}$ is responsible for a change in phase of the wave, which moves it in space. We can extract this imaginary term that is responsible for the phase change from

$$a\Delta t = \ln(1 - \sigma + \sigma e^{-i\kappa_m\Delta x}), \quad (10.13)$$

and then extract the imaginary components

$$\Im(a\Delta t) = \Im[\ln(1 - \sigma + \sigma e^{-i\kappa_m\Delta x})]. \quad (10.14)$$

In the ideal case, this should be identical to the exact dispersion relation, but in practical applications, it is typically either slower or faster, resulting in some phase error from the wave moving at an approximate speed. We can extract the relative speed from

$$\frac{\Im(a\Delta t)}{-i\alpha\kappa_m\Delta t} = \frac{\Im[\ln(1 - \sigma + \sigma e^{-i\kappa_m\Delta x})]}{-i\alpha\kappa_m\Delta t}, \quad (10.15)$$

which tells us the speed of the numerical wave relative to the exact wave speed. When this value is less than 1 the wave moves too slowly, and when this value is greater than 1 the wave moves too fast.



Check out the Von Neumann Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

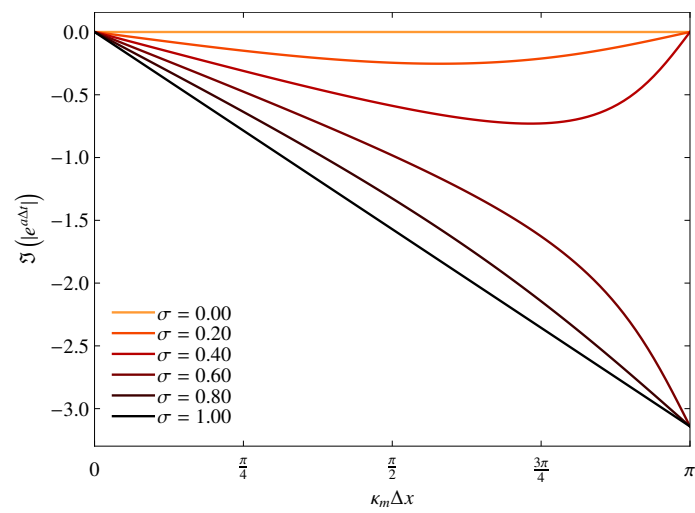


Figure 10.4: Dispersion curve for the implicit advection scheme

11. Modified Equation Analysis

In the previous sections, we have demonstrated that our numerical schemes only approximate the exact PDE we are trying to solve. Furthermore, via von Neumann analysis, we can quantify the type of error we will observe as either dissipation or dispersion. Modified equation analysis is another powerful technique that allows us to better understand how and why a particular numerical scheme behaves the way it does. Using modified equation analysis, we can show that, while our numerical scheme does not exactly satisfy the PDE we are trying to solve, it does provide an exact solution to a similar PDE. By determining what this similar PDE is, we can gain further insight into the behaviour of our scheme. In this section, we will demonstrate by example how to perform modified equation analysis in the context of linear advection.

11.1 Linear Advection

Here, we will derive the modified equation for our first order linear advection scheme

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} + \alpha \frac{u_i^t - u_{i-1}^t}{\Delta x} = 0. \quad (11.1)$$

In order to derive the modified equation, we will start by re-stating our Taylor-Series expansions for the solution at each grid point and time level, dropping the evaluated at notation for compactness and expanding up to the second-order terms, as

$$u_{i-1}^t = u_i^t - \Delta x \frac{\partial u}{\partial x} + \frac{\Delta x^2}{2} \frac{\partial^2 u}{\partial x^2} + \mathcal{O}(\Delta x^3), \quad (11.2)$$

and using a similar Taylor-Series in time

$$u_i^{t+1} = u_i^t - \Delta t \frac{\partial u}{\partial t} + \frac{\Delta t^2}{2} \frac{\partial^2 u}{\partial t^2} + \mathcal{O}(\Delta t^3). \quad (11.3)$$

We will now insert these expansions into our numerical scheme for their respective terms

$$\frac{u_i^t - \Delta t \frac{\partial u}{\partial t} + \frac{\Delta t^2}{2} \frac{\partial^2 u}{\partial t^2} + \mathcal{O}(\Delta t^3) - u_i^t}{\Delta t} + \alpha \frac{u_i^t - u_i^t - \Delta x \frac{\partial u}{\partial x} + \frac{\Delta x^2}{2} \frac{\partial^2 u}{\partial x^2} + \mathcal{O}(\Delta x^3)}{\Delta x} = 0. \quad (11.4)$$

Simplifying this expression down a bit yields

$$-\frac{\partial u}{\partial t} + \frac{\Delta t}{2} \frac{\partial^2 u}{\partial t^2} + \mathcal{O}(\Delta t^2) + \alpha \left[-\frac{\partial u}{\partial x} + \frac{\Delta x}{2} \frac{\partial^2 u}{\partial x^2} + \mathcal{O}(\Delta x^2) \right] = 0, \quad (11.5)$$

which can then be rearranged to

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = -\frac{\Delta t}{2} \frac{\partial^2 u}{\partial t^2} + \alpha \frac{\Delta x}{2} \frac{\partial^2 u}{\partial x^2} + \mathcal{O}(\Delta x^2, \Delta t^2). \quad (11.6)$$

We can recognize the right-hand side of this equation as the truncation error of our scheme and we note that in the limit as Δt and Δx go to zero this will converge to the exact PDE. Furthermore, since the behaviour of the temporal derivative on the left-hand side is somewhat unclear, we will convert it to a spatial derivative. Starting by differentiating the above expression in time, we get

$$\frac{\partial^2 u}{\partial t^2} + \alpha \frac{\partial^2 u}{\partial x \partial t} = -\frac{\Delta t}{2} \frac{\partial^3 u}{\partial t^3} + \alpha \frac{\Delta x}{2} \frac{\partial^3 u}{\partial x^2 \partial t} + \mathcal{O}(\Delta x^2, \Delta t^2), \quad (11.7)$$

and in space, we get

$$\frac{\partial^2 u}{\partial t \partial x} + \alpha \frac{\partial^2 u}{\partial x^2} = -\frac{\Delta t}{2} \frac{\partial^3 u}{\partial t^2 \partial x} + \alpha \frac{\Delta x}{2} \frac{\partial^3 u}{\partial x^3} + \mathcal{O}(\Delta x^2, \Delta t^2). \quad (11.8)$$

Rearranging and substituting Equation 11.8 into Equation 11.7 yields

$$\frac{\partial^2 u}{\partial t^2} = \alpha^2 \frac{\partial^2 u}{\partial x^2} + \mathcal{O}(\Delta x, \Delta t). \quad (11.9)$$

Hence, if we go back to Equation 11.10 we can replace our second derivative in time with a second derivative in space, yielding

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = -\frac{\Delta t}{2} \alpha^2 \frac{\partial^2 u}{\partial x^2} + \alpha \frac{\Delta x}{2} \frac{\partial^2 u}{\partial x^2} + \mathcal{O}(\Delta x^2, \Delta t^2, \Delta x \Delta t^2, \Delta t^3). \quad (11.10)$$

Then, using the definition of the CFL number previously defined as $\sigma = \alpha \Delta t / \Delta x$

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = \frac{\alpha \Delta x}{2} (1 - \sigma) \frac{\partial^2 u}{\partial x^2} + \mathcal{O}(\Delta x^2, \Delta t^2, \Delta x \Delta t^2, \Delta t^3), \quad (11.11)$$

and rearranging finally yields

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} - \frac{\alpha \Delta x}{2} (1 - \sigma) \frac{\partial^2 u}{\partial x^2} = \mathcal{O}(\Delta x^2, \Delta t^2, \Delta x \Delta t^2, \Delta t^3). \quad (11.12)$$

If we look at the form of the Equation 11.12, we see that when Δx and Δt are small the right-hand side reduces to zero, and we are left with a general *advection-diffusion* equation. Hence, we have shown that our finite difference scheme is actually the exact solution to an advection-diffusion problem, rather than the linear advection equation we initially intended to solve. Furthermore, we note that the diffusion operator will only be valid for $0 \leq \sigma \leq 1$, after which point the sign of the term will change and it will become a non-physical *anti-diffusive* operator. This switch from a physical to non-physical PDE coincides with the stability limits we had originally derived for this scheme using von Neumann analysis. Also, as we would expect, our solutions using this method also behave exactly like an advection-diffusion equation since the solution also move, but simultaneously dissipates. Also, for the value $\sigma = 1$, this diffusion operator disappears, which is also consistent with our observations and predictions based on von Neumann analysis, which showed that the amplification factor $|e^{a\Delta t}| = 1$ for this case.

11.2 General Observations

It is important to note that the procedure followed here for the first order linear advection scheme can be repeated for any finite difference scheme of your choice. Starting with your numerical scheme, you can replace each term with its Taylor Series expansion and rearrange this to try to get the PDE of interest on the left-hand side and truncation error on the right-hand side. The leading order terms will tell you the nature of the dominant error in the scheme. If the dominant error has an even-order derivative, such as the scheme above, then one would expect that the scheme is *dominantly dissipative*. In contrast, if the dominant error has an odd-order derivative, then one would expect that the scheme is *dominantly dispersive*. Hence, similar to von Neumann analysis, modified equation analysis is a powerful tool to aid in understanding the general behaviour of a numerical scheme, to elucidate its dissipative and dispersive error properties, and to identify its stability limits.

12. Time-Stepping

In the previous sections, we have always approximated the time derivative using first-order finite differences, assuming

$$\frac{\partial u}{\partial t} = \frac{u_i^{t+1} - u_i^t}{\Delta t} + \mathcal{O}(\Delta t). \quad (12.1)$$

In the von Neumann analysis section, we demonstrated that two linear advection schemes and two linear diffusion schemes are either conditionally or unconditionally stable using this approach. However, if we try to use this finite difference approach to get higher-order accurate schemes in time, such as a central in time approach

$$\frac{\partial u}{\partial t} = \frac{u_i^{t+1} - u_i^{t-1}}{2\Delta t} + \mathcal{O}(\Delta t^2), \quad (12.2)$$

we find it is almost always *unstable*. Hence, getting higher than first-order accurate solutions in time requires something other than finite differences. Furthermore, for both of the unconditionally stable schemes in the von Neumann section, we have not yet discussed how to actually advance them in time, and how to get higher-order accuracy in time, while maintaining this unconditional stability. Hence, in this section, we will discuss more appropriate discretizations for the time derivative.

12.1 Explicit

Our previous approach for discretizing the time derivative is called a *multi-step* scheme since we are using information from multiple time steps to try to estimate the solution at the next time-step. In this section we will introduce *multi-stage* time stepping, specifically the well-known classical Runge-Kutta methods. Rather than use the value of the solution at previous time steps, Runge-Kutta methods compute the solution at intermediate stages between time level t and $t + 1$. Then, these intermediate solutions are cleverly combined to achieve a more stable and potentially higher-order accurate solution in time. In all of these cases, we will re-arrange our system into a general form

$$\frac{\partial u}{\partial t} = R(u), \quad (12.3)$$

where $R(u)$ is typically referred to as the *residual* or the *right-hand side*. Note that the term residual is used in several different contexts in CFD.

12.1.1 Forward Euler

The forward Euler, or explicit Euler, scheme uses just one stage to predict the solution at the next time step. In fact, it is equivalent to our previous finite difference approach. In this scheme we get

$$u^{t+1} = u^t + \Delta t R(u). \quad (12.4)$$

Graphically, this can be interpreted as using the current derivative of the solution in time at time t , and simply extrapolating based on this derivative to the time $t + 1$. This approach is $\mathcal{O}(\Delta t)$ in time.

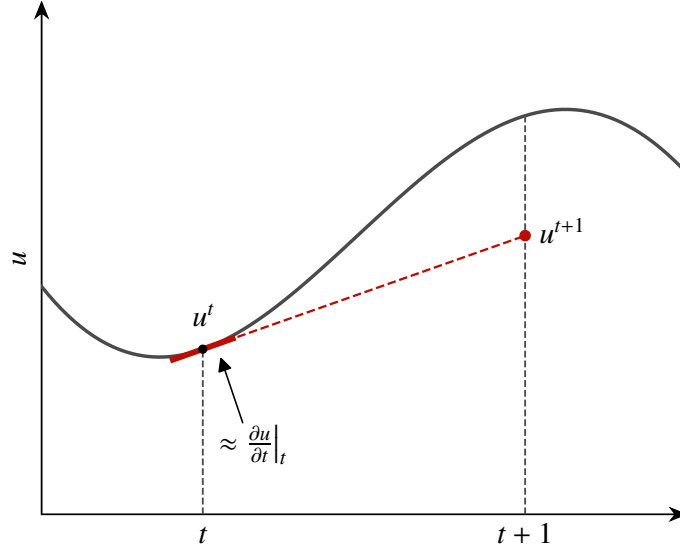


Figure 12.1: Explicit Euler

12.1.2 Heun's Method

Heun's method is a fairly simple but clever improvement on the explicit Euler scheme. We start by using the explicit Euler method to build an approximation of the solution at $t + 1$

$$\tilde{u} = u^t + \Delta t R(u), \quad (12.5)$$

where \tilde{u} is an initial guess for the solution. Then, we use an average of the following right-hand sides to compute the solution at the next time step

$$\tilde{u}^{t+1} = u^t + \Delta t \left[\frac{R(u) + R(\tilde{u})}{2} \right]. \quad (12.6)$$

This approach is called a two-stage scheme, since we had to compute our solution at one intermediate stage before the final solution is obtained. This requires us to evaluate two right-hand sides, one for each stage, so is approximately twice as expensive as the explicit Euler method. However, it can be shown that it achieves $\mathcal{O}(\Delta t^2)$ in time. Hence, when the time step is small, it is significantly more accurate.

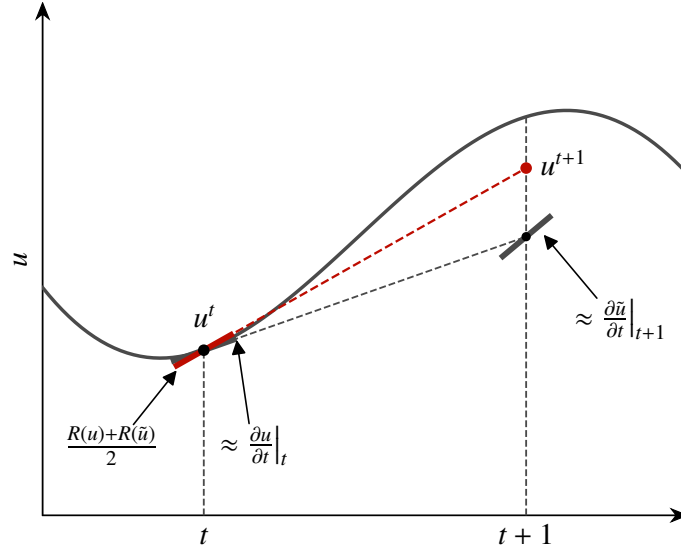


Figure 12.2: Heun's method

12.1.3 Midpoint Method

The midpoint method is very similar to Heun's method. However, we start by computing the solution at an intermediate stage halfway between the current and next time steps, using an explicit Euler approach

$$\tilde{u} = u^t + \frac{\Delta t}{2} R(u). \quad (12.7)$$

Since we have divided Δt by two, we are now halfway between the current and next time steps, hence why it is called the midpoint method. Now, we compute the solution at the next time step by evaluating the residual at this midpoint location

$$u^{t+1} = u^t + \Delta t R(\tilde{u}). \quad (12.8)$$

Similar to Heun's method, this is also a two stage scheme and has a temporal error of $\mathcal{O}(\Delta t^2)$.

12.1.4 Runge-Kutta Methods

If we look at Heun's method and the midpoint method, we note that their structure is very similar. We start by computing some intermediate solution, then evaluating the residual of that intermediate solution, and then linearly combining it with the residual of the initial solution. This can be generalized from two stages up to as many stages as we may like, with multiple intermediate solutions being obtained to improve the order of accuracy. Starting from our current solution u^t , we will compute s intermediate stage solutions

$$u_1, u_2, \dots, u_s, \quad (12.9)$$

where the subscript of u_i denotes the stage number and s is the total number of stages. From these, we can also get the residuals of these intermediate solutions from $R(u_i)$. We compute the intermediate stage solutions from

$$u_i = u^t + \Delta t \sum_{j=1}^s a_{i,j} R(u_j), \quad (12.10)$$

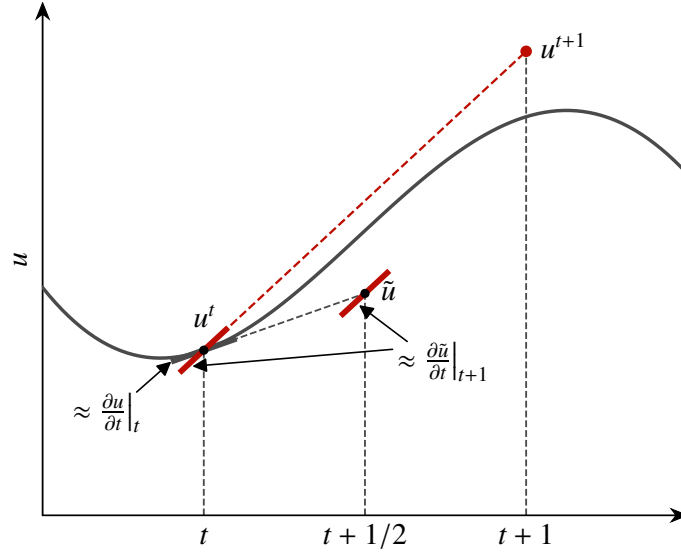


Figure 12.3: Midpoint method

and the final solution is found from

$$u^{t+1} = u^t + \Delta t \sum_{i=1}^s b_i R(u_i), \quad (12.11)$$

where $a_{i,j}$ and b_i and a set of constants. These can be compacted into a *Butcher Tableau*

$$\begin{array}{c|cc} c & A & \\ \hline & \mathbf{b} & \end{array} = \begin{array}{c|ccc} c_1 & a_{1,1} & \cdots & a_{1,s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s,1} & \cdots & a_{s,s} \\ \hline & b_1 & \cdots & b_s \end{array} \quad (12.12)$$

This will yield an explicit scheme if the A matrix is strictly lower-triangular. In that case, we find the solution at any stage is simple a function of the solution of earlier stages. In that case, the method consists of a number of intermediate stage solutions computed using an explicit Euler approach. The following are some example tableaus for commonly-used explicit schemes, including those already introduced.

Explicit Euler

The explicit Euler method has one stage and is $\mathcal{O}(\Delta t)$ in time.

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \quad (12.13)$$

Heun's Method

Heun's method has two stages and is $\mathcal{O}(\Delta t^2)$ in time.

$$\begin{array}{c|cc} 0 & & \\ \hline 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \quad (12.14)$$

Explicit Midpoint

The explicit midpoint method has two stages and is $\mathcal{O}(\Delta t^2)$ in time.

$$\begin{array}{c|cc} 0 & & \\ \hline \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array} \quad (12.15)$$

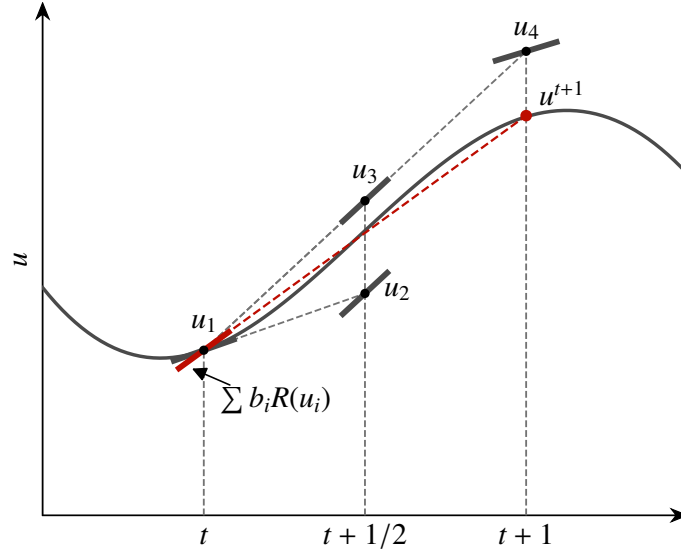


Figure 12.4: Fourth-order four-stage RK method

Fourth-Order Runge-Kutta

The explicit four-stage fourth-order Runge-Kutta method is $\mathcal{O}(\Delta t^4)$ in time.

$$\begin{array}{c|ccc}
 0 & & & \\
 \frac{1}{2} & \frac{1}{2} & & \\
 \frac{1}{2} & 0 & \frac{1}{2} & \\
 1 & 0 & 0 & 1 \\
 \hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
 \end{array} \tag{12.16}$$

12.2 Implicit

Going back to the von Neumann analysis section, we note that implicit schemes, those where we evaluated the spatial derivative using the solution at the as-yet-unknown next time step, were unconditionally stable for both linear advection and diffusion. For example, the following scheme was introduced for the linear advection equation

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} + \alpha \frac{u_i^{t+1} - u_{i-1}^{t+1}}{\Delta x} = 0, \tag{12.17}$$

and the following for the linear diffusion equation

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} - \beta \frac{u_{i-1}^{t+1} - 2u_i^{t+1} + u_{i+1}^{t+1}}{\Delta x^2} = 0. \tag{12.18}$$

These are very similar to our initial schemes, except the spatial operators are evaluated at $t + 1$ rather than t .

12.2.1 Implicit Linear Advection

We will start our exploration of implicit schemes with the linear advection equation, and rearrange it so all of the terms involving the unknown solution at $t + 1$ are of the left-hand side and all of the terms involving the current solution at time t are on the right-hand side

$$u_i^{t+1} + \sigma (u_i^{t+1} - u_{i-1}^{t+1}) = u_i^t, \tag{12.19}$$

where as usual $\sigma = \alpha \Delta t / \Delta x$. Then, lumping the terms for each grid point on the left-hand side yields

$$(1 + \sigma)u_i^{t+1} - \sigma u_{i-1}^{t+1} = u_i^t. \quad (12.20)$$

Unlike the explicit schemes, where we could readily rearrange the expression to get u_i^{t+1} alone on the left-hand side, we now have one equation for each grid point that has two unknowns, u_i^{t+1} and u_{i-1}^{t+1} , on the left-hand side. This means that in order to get u_i^{t+1} we need to already know u_{i-1}^{t+1} , and in order to get u_{i-1}^{t+1} we need to already know u_{i-2}^{t+1} , and so on. Hence, it appears we are stuck in a situation where we need to already know the solution at every grid point to get the solution at every other grid point, which we don't yet have.

To get around this, we can recognize that we actually have a large system of linear equations. For each of the N grid points, there is an expression of the form of Equation 12.20, and these involve some linear combination of the N unknown solution values at $t + 1$. Hence, we have N equations and N unknowns, yielding a linear system to be solved. This can be written as a linear system of the form

$$A\vec{u}^{t+1} = \vec{u}^t, \quad (12.21)$$

where, for the case of periodic boundary conditions

$$A = \begin{bmatrix} 1 + \sigma & 0 & 0 & \dots & \sigma \\ -\sigma & 1 + \sigma & 0 & \dots & 0 \\ 0 & -\sigma & 1 + \sigma & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -\sigma & 1 + \sigma \end{bmatrix}, \quad (12.22)$$

is referred to as the Jacobian matrix, and

$$\vec{u}^t = \begin{bmatrix} u_1^t \\ u_2^t \\ u_3^t \\ \vdots \\ u_N^t \end{bmatrix}, \quad (12.23)$$

is a vector of the solution at all grid points at the current time step, and

$$\vec{u}^{t+1} = \begin{bmatrix} u_1^{t+1} \\ u_2^{t+1} \\ u_3^{t+1} \\ \vdots \\ u_N^{t+1} \end{bmatrix}, \quad (12.24)$$

is a vector of the unknown solution values at the next time step, which is what we are trying to find. Note that the structure of the Jacobian matrix arises directly from writing out the equation for each grid point, and then assembling that as a linear system of equations. It is clear from Equation 12.21 that we can now solve for all of the solution values in \vec{u}^{t+1} by simply solving a linear system of equations. Hence, we obtain unconditional stability, as discussed in the von Neumann analysis section, at the added cost of having to solve a linear system of equations at each time step. This cost is usually substantial and, hence, implicit schemes are typically used when sufficiently large time steps that are larger than the stability limits of explicit schemes are desired.

12.2.2 Implicit Linear Diffusion

As a second demonstration of an implicit solver, we consider the implicit linear diffusion equation, derived previously as

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} - \beta \frac{u_{i-1}^{t+1} - 2u_i^{t+1} + u_{i+1}^{t+1}}{\Delta x^2} = 0. \quad (12.25)$$

Rearranging this so that all of the known solution values are on the right-hand side and all values at the as yet unknown next time step are on the left-hand side yields

$$u_i^{t+1} - r(u_{i-1}^{t+1} - 2u_i^{t+1} + u_{i+1}^{t+1}) = u_i^t, \quad (12.26)$$

where again $r = \beta \Delta t / \Delta x^2$. Rearranging slight yields

$$(1 + 2r)u_i^{t+1} - r(u_{i-1}^{t+1} + u_{i+1}^{t+1}) = u_i^t. \quad (12.27)$$

Following the same steps as the linear advection equation, this can be written as a linear system of equations of the form

$$A\vec{u}^{t+1} = \vec{u}^t, \quad (12.28)$$

where the Jacobian matrix for periodic boundary conditions can be written as

$$A = \begin{bmatrix} 1+2r & -r & 0 & \dots & -r \\ -r & 1+2r & -r & \dots & 0 \\ 0 & -r & 1+2r & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -r & 0 & \dots & -r & 1+2r \end{bmatrix}, \quad (12.29)$$

taking note of the locations of the $-r$ terms in the first and last rows that arise from the periodic boundary conditions. Hence, similar to the implicit linear advection scheme, implicit linear diffusion requires the solution of a linear system of equations at each time step. Therefore, it is typically used when the desired time step exceeds the stability limit of available explicit methods. Due to the Δx^2 scaling of the time step size for diffusion equations, as discussed in the von Neumann analysis section, diffusion operators are often solved using implicit approaches.

12.2.3 Implicit Burgers Equation

In both of the previous sections, we looked at implicit solvers for *linear* systems of equations, specifically the linear advection and linear diffusion equations. In this section, we will consider an implicit solver for the *non-linear* Burgers equation with a numerical scheme of the form

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} + \frac{1}{2\Delta x} \left((u_i^{t+1})^2 - (u_{i-1}^{t+1})^2 \right) = 0. \quad (12.30)$$

Starting from a suitable initial guess for \vec{u}^{t+1} , which we will denote as \vec{u}^k where k here denotes an iteration number, we can compute the resulting residual at any gridpoint as

$$r_i = \frac{u_i^k - u_i^t}{\Delta t} + \frac{1}{2\Delta x} \left((u_i^k)^2 - (u_{i-1}^k)^2 \right). \quad (12.31)$$

From this we can compute the unsteady residual array $\vec{r}(\vec{u}^k)$, which determines how well our initial guess satisfies the system of equations. Our objective is to find a value of \vec{u}^k such that this residual is zero. Once this is achieved we have found a solution and we can take $\vec{u}^{t+1} = \vec{u}^k$.

One of the most powerful approaches to solving this is to use Newton-Raphson. Starting with \vec{u}^k , our current guess of the solution we will iterate by attempting to enforce that the next iterated values satisfies

$$\vec{r}(\vec{u}^{k+1}) = 0. \quad (12.32)$$

Via the application of Newton-Raphson we obtain the following expression

$$\vec{u}^{k+1} = \vec{u}^k - \left(\frac{\partial \vec{r}}{\partial \vec{u}^k} \right)^{-1} \vec{r}(\vec{u}^k), \quad (12.33)$$

where the Jacobian matrix, for the current Burgers equation discretization, is given by

$$\frac{\partial \vec{r}}{\partial \vec{u}^k} = \begin{bmatrix} 1 + \frac{\Delta t}{\Delta x} u_i^k & 0 & 0 & \dots & -\left(\frac{\Delta t}{\Delta x} u_{i-1}^k\right) \\ -\left(\frac{\Delta t}{\Delta x} u_{i-1}^k\right) & 1 + \frac{\Delta t}{\Delta x} u_i^k & 0 & \dots & 0 \\ 0 & -\left(\frac{\Delta t}{\Delta x} u_{i-1}^k\right) & 1 + \frac{\Delta t}{\Delta x} u_i^k & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -\left(\frac{\Delta t}{\Delta x} u_{i-1}^k\right) & 1 + \frac{\Delta t}{\Delta x} u_i^k \end{bmatrix}. \quad (12.34)$$

Hence, the general procedure for solving nonlinear systems of equations is to start by guessing a solution \vec{u}^{t+1} , which is stored as \vec{u}^k . Typically, a good initial guess is that $\vec{u}^k = \vec{u}^t$. Using this initial guess, the residual and Jacobian matrices are computed and, via application of Newton-Raphson, an improved guess \vec{u}^{k+1} is obtained. This procedure is then repeated until $\vec{r}(\vec{u}^k) \approx 0$, at which point we take $\vec{u}^{t+1} = \vec{u}^k$. In the case of linear systems, only one Newton iterations is required. However, for non-linear systems such as Burgers or Navier-Stokes, several Newton iterations are typically required for sufficient convergence.

13. Iterative Methods

In this implicit time stepping section we reduced each time step to the solution of a linear system of N equations of the form

$$A\vec{x} = \vec{b}, \quad (13.1)$$

when solving either linear advection or linear diffusion, where N is the number of grid points in the domain. The extra cost of solving this linear system was introduced in order to obtain unconditionally stable schemes, allowing very large time steps to be taken. However, this will only be faster than an explicit approach if the linear system of equations can be solved efficiently. Hence, this section is dedicated to exploring different methods of solving linear systems of equations.

13.1 Gaussian Elimination

Perhaps the most straightforward method for solving a linear system of equations is Gaussian elimination. This requires us to invert A and obtain

$$\vec{x} = A^{-1}\vec{b}. \quad (13.2)$$

Gaussian elimination is the workhorse of most undergraduate linear algebra courses, and is a straightforward extension of the substitution approach for solving systems of equations taught in highschool. This requires inverting A or an LU decomposition of A and \vec{b} to be formed, and then back substitution to solve for the unknown components of \vec{x} .

While Gaussian elimination is well known, its computational cost scales like $\mathcal{O}(N^3)$ where N is the number of unknowns. Hence, as the system of equations, and in our case number of grid points, grows, the computational cost of Gaussian elimination grows cubically. Any undergraduate student should be familiar with trying to solve even 3×3 linear systems in an exam, let alone 4×4 , or 5×5 . With each additional equation, doing this by hand takes significantly more time. Hence, in the context of CFD, where thousands or millions of equations is common practice, Gaussian elimination is rarely used.

13.2 Jacobi Iteration

Since the formation of A^{-1} is prohibitively expensive for large systems of equations, we will try to find an approximate solution \vec{x} , without actually forming A^{-1} . This can be done iteratively, and the first approach we will introduce is Jacobi iteration. In Jacobi iteration, we start by splitting A into L , D , and U , and are its strictly lower-triangular, diagonal, and strictly upper-triangular components. This allows us to write our linear system as

$$(L + D + U)\vec{x} = \vec{b}, \quad (13.3)$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}, \quad (13.4)$$

$$L = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ a_{21} & 0 & 0 & \dots & 0 \\ a_{31} & a_{32} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & 0 \end{bmatrix}, \quad (13.5)$$

$$D = \begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 \\ 0 & a_{22} & 0 & \dots & 0 \\ 0 & 0 & a_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a_{nn} \end{bmatrix}, \quad (13.6)$$

$$U = \begin{bmatrix} 0 & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & 0 & a_{23} & \dots & a_{2n} \\ 0 & 0 & 0 & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}. \quad (13.7)$$

Now if \vec{x} is a solution to the linear system then the following is valid

$$L\vec{x} + D\vec{x} + U\vec{x} = \vec{b}. \quad (13.8)$$

We can now rearrange this expression

$$D\vec{x} = \vec{b} - (L + U)\vec{x}, \quad (13.9)$$

and by inverting D we get

$$\vec{x} = D^{-1} [\vec{b} - (L + U)\vec{x}]. \quad (13.10)$$

At this point, it is important to know that while inverting A is very expensive, inverting just its diagonal values in D is trivial.

We note that if \vec{x} is a solution to the linear system of equations, then Equation 13.10 will be satisfied exactly, but if we already had the exact solution there would be no point in this exercise. However, if we look again at Equation 13.10 it has an interesting form. We note that if we insert some guess for \vec{x} into the right-hand side, we get out a modified \vec{x} on the left-hand side. This allows us to define Jacobi iteration as

$$\vec{x}_{n+1} = D^{-1} \left[\vec{b} - (L + U)\vec{x}_n \right], \quad (13.11)$$

where \vec{x}_n is an approximate solution, and \vec{x}_{n+1} is an updated approximation. Then we can simply pass \vec{x}_{n+1} back through this formulation to obtain \vec{x}_{n+2} , and so on, iterating to a final solution when the value of \vec{x} converges. This can also be written in an element-by-element manner as

$$x_i^{n+1} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j \neq i} a_{i,j} x_j^n \right), \quad i = 1, 2, \dots, n. \quad (13.12)$$

Each step of Jacobi iteration requires $\mathcal{O}(N^2)$ operations, hence, we can expect Jacobi to outperform Gaussian elimination, as long as we need fewer iterations than there are rows in the system of equations. Furthermore, in CFD it is usually sufficient to converge \vec{x} to some finite level of precision.

To demonstrate the utility of Jacobi iteration we will use a simple example of a 3×3 system of equations. Starting with

$$A = \begin{bmatrix} 10 & 2 & 4 \\ 6 & 8 & 4 \\ 2 & 3 & 9 \end{bmatrix}, \quad (13.13)$$

and

$$\vec{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad (13.14)$$

we obtain the following for L , D , and U

$$L = \begin{bmatrix} 0 & 0 & 0 \\ 6 & 0 & 0 \\ 2 & 3 & 0 \end{bmatrix}, \quad (13.15)$$

$$D = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 9 \end{bmatrix}, \quad (13.16)$$

$$U = \begin{bmatrix} 0 & 2 & 4 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix}. \quad (13.17)$$

Furthermore, we can easily invert D to get

$$D^{-1} = \begin{bmatrix} \frac{1}{10} & 0 & 0 \\ 0 & \frac{1}{8} & 0 \\ 0 & 0 & \frac{1}{9} \end{bmatrix}. \quad (13.18)$$

Now we can start with some initial guess, lets say

$$\vec{x}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \quad (13.19)$$

Convergence can usually be accelerated by using a good initial guess, for example, that the solution at the next time step in our CFD simulation is the same as the current solution.

Now with our initial Guess and all terms from the right-hand side of Equation 13.11 defined, we can compute

$$\vec{x}_1 = D^{-1} \left[\vec{b} - (L + U)\vec{x}_0 \right] = \begin{bmatrix} 0.1000 \\ 0.2500 \\ 0.3333 \end{bmatrix}. \quad (13.20)$$

Repeating this iteration a few times yields

$$\vec{x}_2 = D^{-1} \left[\vec{b} - (L + U)\vec{x}_1 \right] = \begin{bmatrix} -0.0833 \\ 0.0083 \\ 0.2277 \end{bmatrix}, \quad (13.21)$$

$$\vec{x}_3 = D^{-1} \left[\vec{b} - (L + U)\vec{x}_2 \right] = \begin{bmatrix} 0.0072 \\ 0.1986 \\ 0.3490 \end{bmatrix}, \quad (13.22)$$

and after 15 iterations we obtain

$$\vec{x}_{15} = D^{-1} \left[\vec{b} - (L + U)\vec{x}_{14} \right] = \begin{bmatrix} -0.0465 \\ 0.1356 \\ 0.2984 \end{bmatrix}, \quad (13.23)$$

which is very close to the exact solution

$$\vec{x} = \begin{bmatrix} -0.0465 \\ 0.1357 \\ 0.2984 \end{bmatrix}. \quad (13.24)$$

Hence, after only 15 iterations Jacobi was able to converge to approximately four digits, without having to ever directly invert the A matrix.



Check out the Iterative Methods Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

13.3 Gauss Seidel Iteration

With the idea of Jacobi iteration outlined above, we may wonder whether similar more efficient approaches exist. The first of these, with a very similar derivation to Jacobi, is the Gauss-Seidel method. Starting from our split A matrix in the Jacobi section

$$(L + D + U)\vec{x} = \vec{b}, \quad (13.25)$$

we rearrange it as

$$(L + D)\vec{x} = \vec{b} - U\vec{x}. \quad (13.26)$$

If we consider the $(L + D)$ matrix we note that it is lower-triangular. While not as trivial to invert as D in Jacobi iteration, inverting $(L + D)$ can be done quickly via back-substitution. Hence, we can write

$$\vec{x} = (L + D)^{-1} \left[\vec{b} - U\vec{x} \right]. \quad (13.27)$$

Similar to Jacobi, we notice that if we insert an estimate for \vec{x} on the right-hand side, we obtain an updated estimate on the left-hand side. Hence, we define Gauss-Seidel iteration as

$$\vec{x}_{n+1} = (L + D)^{-1} [\vec{b} - U\vec{x}_n]. \quad (13.28)$$

Also, similar to Jacobi this can be performed in a element-wise manner for each component of the solution via

$$x_i^{n+1} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{n+1} - \sum_{j=i+1}^n a_{i,j} x_j^n \right), \quad i = 1, 2, \dots, n. \quad (13.29)$$

It is clear from this that Gauss-Seidel amounts to simply a Jacobi iteration but using the most up to date entries $a_{i,j} x_j^n$ or $a_{i,j} x_j^{n+1}$ as they are available. Usually, the additional cost of computing $(L + D)^{-1}$, relative to simply computing D^{-1} with Jacobi, significantly reduces the number of iterations and total computational cost.

Going back to our example from Jacobi iteration, we have Now with our initial Guess and all terms from the right-hand side of Equation 13.11 defined, we can compute

$$\vec{x}_1 = (L + D)^{-1} [\vec{b} - U\vec{x}_0] = \begin{bmatrix} 0.1000 \\ 0.1750 \\ 0.2527 \end{bmatrix}. \quad (13.30)$$

Repeating this iteration a few times yields

$$\vec{x}_2 = (L + D)^{-1} [\vec{b} - U\vec{x}_1] = \begin{bmatrix} -0.0361 \\ 0.1506 \\ 0.2911 \end{bmatrix}, \quad (13.31)$$

$$\vec{x}_3 = (L + D)^{-1} [\vec{b} - U\vec{x}_2] = \begin{bmatrix} -0.0465 \\ -0.0467 \\ 0.2982 \end{bmatrix}, \quad (13.32)$$

and after only 5 iterations we obtain

$$\vec{x}_5 = (L + D)^{-1} [\vec{b} - U\vec{x}_4] = \begin{bmatrix} -0.0465 \\ 0.1358 \\ 0.2984 \end{bmatrix}, \quad (13.33)$$

which is very close to the exact solution

$$\vec{x} = \begin{bmatrix} -0.0465 \\ 0.1357 \\ 0.2984 \end{bmatrix}. \quad (13.34)$$

Hence, after only 5 iterations, or three times as fast as Jacobi in this case, Gauss Seidel was able to converge to approximately four digits.



Check out the Iterative Methods Jupyter notebook [here](#). You can also download the files from the Gitlab repository [here](#).

13.4 Successive Over-Relaxation

When using the Gauss-Seidel approach, we often observe that the iterated solutions converge gradually to the exact solution for \vec{x} . That is, each iteration takes a step towards the exact solution in a somewhat uniform manner. We can exploit this by simply extrapolating each iteration to move a bit further towards the exact solution, which is known as Successive Over-Relaxation (SOR). This is usually done to accelerate convergence, but as will be discussed later, it can be used to also stabilize convergence.

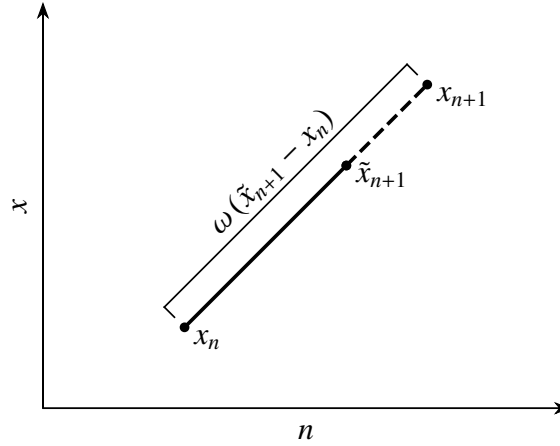


Figure 13.1: Successive Over-Relaxation method.

With SOR we simply apply a Gauss Seidel iteration to our current approximation of the solution, \vec{x}_n . We store the results of this Gauss-Seidel iteration temporarily as $\tilde{\vec{x}}_{n+1}$. Then, using the SOR approach, we linearly project our updated solution using our current and previous approximations, as shown in Figure 13.1. Hence,

$$\vec{x}_{n+1} = \omega \tilde{\vec{x}}_{n+1} + (1 - \omega) \vec{x}_n. \quad (13.35)$$

where ω is referred to as the relaxation factor. We note that when $\omega = 1$ we recover the original Gauss Seidel approach. When $\omega > 1$ we say it is over-relaxed, usually accelerating convergence but also potentially causing the iterations to diverge. When $\omega < 1$ we say it is under-relaxed, which usually improves stability, but reduces the rate of convergence. In general, a suitable value for the relaxation factor is within the range

$$0 \leq \omega \leq 2, \quad (13.36)$$

and values of $\omega > 2$ will diverge. Typically, ω is chosen to be as large as possible, without causing the iterations to diverge.

13.5 Assessing Convergence

In typical CFD applications, the user provides a desired convergence tolerance for solving the linear system of equations. To assess this, we introduce the concept of a residual. In the case of an exact solution to a linear system, we would expect

$$A\vec{x} - \vec{b} = 0. \quad (13.37)$$

However, since our solution at any given iteration is only an approximate solution then

$$A\vec{x}_n - \vec{b} = \vec{r}, \quad (13.38)$$

where \vec{r} is referred to as the residual and measures how well our current approximate solution satisfies the system of equations being solved. While \vec{r} gives detailed information about how well each equation in the entire system is being solved, it is typically more useful to provide the user with a norm, such as $\|\vec{r}\|_2$ or $\|\vec{r}\|_\infty$, giving a single measure of convergence. Then, iterations are continued until this residual norm converges to within the desired tolerance. It is also important to note that if the convergence tolerance is too high, it can result in a loss of accuracy or stability in the CFD simulation. Hence, particular care must be chosen in how to select the desired convergence tolerance.

13.6 Multigrid

It is important to note that both Jacobi and Gauss-Seidel iterations require $\mathcal{O}(N^2)$ operations per iteration, they will typically be much faster than Gaussian elimination, provided the number of iterations is relatively small. However, the computational cost will still increase rapidly with the number of grid points. Ideally, we would like an iterative method that will converge with $\mathcal{O}(N)$ operations, resulting in a computational cost that varies linearly with the number of grid points.

An important observation regarding the aforementioned iterative methods, such as Jacobi and Gauss-Seidel, is that they tend to converge *high-wavenumber* features, those that span only a few grid points, very quickly. In contrast, *low-wavenumber* features, those that span a large number of grid points, converge relatively slowly. The general idea behind the multigrid method is to use an additional coarse grid to converge these large scale structures. Since this additional grid is coarse, it is relatively cheap per iteration, since it contains significantly fewer grid points and unknowns. Also, the low-wavenumber features on the fine grid appear as higher-wavenumber features on the coarse grid and, hence, they converge quickly there. This procedure can then be extended to a large number of grid levels, resulting in the *multigrid* method.

As an example, we will describe a two-level multigrid method for solving the linear system of equations

$$A_h \vec{x}_h = \vec{b}_h, \quad (13.39)$$

where the subscript h denotes that this is the initial system of equations the on fine grid, denoted by Ω_h . Ultimately, we want to obtain the final solution to this linear system, \vec{x}_h , with as little computational cost as possible. From here, we will recall that the residual on the fine grid can be defined as

$$\vec{r}_h = \vec{b}_h - A_h \vec{x}_{h,n}, \quad (13.40)$$

where $\vec{x}_{h,n}$ is our current iterated estimate of the true solution. We will also define the error on the fine grid as simply the difference between the exact solution and the current iterated solution

$$\vec{e}_{h,n} = \vec{x}_h - \vec{x}_{h,n}. \quad (13.41)$$

Rearranging this expression for the error yields

$$\vec{x}_{h,n} = \vec{x}_h - \vec{e}_{h,n}. \quad (13.42)$$

Substituting this into our expression for the expression for the residual yields

$$\vec{r}_h = \vec{b}_h - A_h(\vec{x}_h - \vec{e}_{h,n}), \quad (13.43)$$

and finally

$$A_h \vec{e}_{h,n} = \vec{r}_h. \quad (13.44)$$

Hence, similar to the solution, the error between the exact and current iterated solution also obeys a linear system of equations. In the multigrid method, it is this error on the fine grid that gets projected to the coarse grid, and that is what is solved. The general multigrid process is as follows:

- *Relax* $A_h \vec{x}_h = \vec{b}_h$ using n iterations of Jacobi or Gauss-Seidel to obtain $\vec{x}_{h,n}$
- *Compute* the residual on the fine grid level using $\vec{r}_h = \vec{b}_h - A_h \vec{x}_{h,n}$
- *Restrict* the residual from $\Omega_h \rightarrow \Omega_{2h}$ using $\vec{r}_{2h} = I_h^{2h} \vec{r}_h$
- *Solve* $A_{2h} \vec{e}_{2h,n} = \vec{r}_{2h}$ using Jacobi, Gauss-Seidel, or Gaussian elimination
- *Prolongate* the error from $\Omega_{2h} \rightarrow \Omega_h$ using $\vec{e}_h = I_{2h}^h \vec{e}_{2h,n}$
- *Correct* the solution on the fine grid using $\vec{x}_{h,n} = \vec{x}_{h,n} + \vec{e}_h$
- *Repeat* from the first step until the system of equations on the fine grid converges

In the above approach, we have introduced two new operators, specifically the restriction operator I_h^{2h} and the prolongation operator I_{2h}^h that are responsible for transferring data from the fine to coarse grid, and vice-versa. If we look at their required shapes, we find that $I_h^{2h} \in N_{2h} \times N_h$, where N_{2h} is the number of degrees of freedom on the coarse grid and N_h is the number of degrees of freedom on the fine grid. In contrast, the prolongation operator is of dimension $I_{2h}^h \in N_h \times N_{2h}$. Consider the one-dimensional example case shown in Figure ??, where Ω_{2h} is coarser than Ω_h in the sense that every other grid point is omitted. One simple option is to simply *linearly* interpolate the error from the coarse grid to the fine grid. This would give the following prolongation operator matrix

$$I_{2h}^h = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{bmatrix}, \quad (13.45)$$

assuming periodic boundary conditions and a linear interpolation between the intermediate points. Now, to get the restriction operator, which transfers data from the fine grid to the coarse grid, we use

$$I_h^{2h} = c(I_{2h}^h)^T, \quad (13.46)$$

where c is a constant chosen such that all row sums of I_h^{2h} are unity. This expands to

$$I_h^{2h} = c \begin{bmatrix} 1 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} \end{bmatrix}, \quad (13.47)$$

and taking $c = 1/2$ yields

$$I_h^{2h} = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}, \quad (13.48)$$

satisfying the condition of having row sums of unity. Now that we have the restriction and prolongation operators, the last step is to find the linear system of equations on the coarse grid. This is found via

$$A_{2h} = I_h^{2h} A_h I_{2h}^h. \quad (13.49)$$

Hence, starting from the linear system of equations and the node locations on the coarse and fine grids, the first step for constructing the system on the coarse grid level is to provide a suitable prolongation operator to take data from this coarse to the fine grid. From this, one obtains the corresponding restriction operator, and finally the operator matrix on the coarse grid. Now, all of the required data is available to complete the multigrid procedure. We note that the above procedure describes a two-level multigrid method. However, this can be readily extended to any number of multigrid stages, and the smallest grid levels are typically coarse enough to be solved directly using Gaussian elimination. Furthermore, these grid levels can be traversed in different orders. For example, the V-cycle, W-cycle, and Full Multigrid cycle shown in Figure 13.2. Typically, each cycle spends the majority of its iterations on the coarsest grid levels, since iterations are relatively inexpensive here. Iterations are only performed on the finest grid level sparingly, and to converge the highest-wavenumber features in the flow.

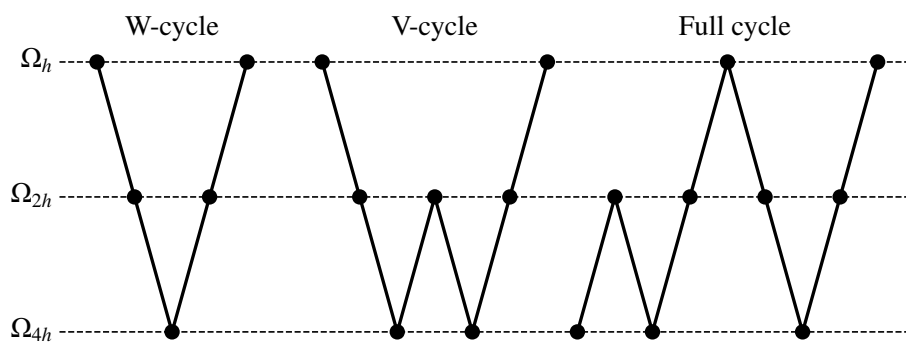


Figure 13.2: Common multigrid methods using three refinement levels

14. Applications

14.1 An Euler Solver

In this section, we will describe how to develop a two-dimensional solver for the Euler equations on a periodic domain. This will be approximated using finite difference methods in space and advanced in time using an explicit Runge-Kutta method.

Note that the Euler equations in two dimensions can be expanded as

$$\frac{\partial \vec{w}}{\partial t} + \frac{\partial \vec{F}}{\partial x} + \frac{\partial \vec{G}}{\partial y} = 0, \quad (14.1)$$

where

$$\vec{w} = [\rho, \rho u, \rho v, \rho E]^T, \quad (14.2)$$

is a vector of the conserved variables, u and v are the velocity components, and E is the specific energy. The fluxes can be found directly as a function of the solution

$$\vec{F} = [\rho u, \rho u^2 + p, \rho uv, u(\rho E + p)]^T, \quad (14.3)$$

and

$$\vec{G} = [\rho v, \rho uv, \rho v^2 + p, v(\rho E + p)]^T, \quad (14.4)$$

where

$$p = \rho RT, \quad (14.5)$$

is the ideal gas law where R is the gas constant and T is temperature. Through some manipulation of the ideal gas law it can be shown that this can be written in an alternative form

$$p = (\gamma - 1)\rho \left(E - \frac{1}{2}(u^2 + v^2) \right), \quad (14.6)$$

that no longer requires the gas constant, and where γ is the ratio of specific heats. Hence, given the value of the solution at any point in the domain, and the ideal gas law, we can determine both of the flux components.

It is worth taking a moment to consider this set of equations. Using the finite difference approach, we will store the conserved variables, specifically mass, momentum, and energy, at each grid point. From Equation 14.15, it is then possible to then get the time derivative of the solution at all of these grid points, provided we can compute the fluxes and their derivatives in space. This simply requires the specification of suitable finite difference methods for approximating the derivatives in space.

Now consider a two-dimensional domain of length L_x and L_y in each direction that is discretized using a grid of N_x and N_y equidistant points. Using periodic boundary conditions, the grid spacing between adjacent points is

$$\Delta x = \frac{L_x}{N_x}, \quad (14.7)$$

in the x-direction and

$$\Delta y = \frac{L_y}{N_y}, \quad (14.8)$$

in the y-direction. The current value of the solution is stored at all grid points and denoted by $\vec{w}_{i,j}^t$ where i and j are the grid point indices and t is the time level. We also store the coordinates of each grid point as $\vec{x}_{i,j}$, where

$$\vec{x}_{i,j} = [x_{i,j}, y_{i,j}]^T. \quad (14.9)$$

Provided an initial condition, the value of the solution is known at each grid point at the start of the simulation. Then the fluxes at each grid point can be calculated according to Equation 14.3 and Equation 14.4. Hence, at every grid point we will now have $\vec{w}_{i,j}^t$, $\vec{F}_{i,j}^t$, and $\vec{G}_{i,j}^t$. Now using, for example, second-order central differences in space, we can write

$$\frac{d\vec{w}_{i,j}^t}{dt} + \frac{\vec{F}_{i+1,j}^t - \vec{F}_{i-1,j}^t}{2\Delta x} + \frac{\vec{G}_{i,j+1}^t - \vec{G}_{i,j-1}^t}{2\Delta y} + \mathcal{O}(\Delta x^2, \Delta y^2) = 0. \quad (14.10)$$

Hence, given a value of the solution at every grid point, the fluxes, their divergence, and finally the time rate of change of the solution at each grid point can be determined. Then, using an appropriate temporal scheme, such as the classical RK_{4,4} method, this system of equations can be advanced to the next time step.

As an example, we will consider advection of an isentropic vortex having the initial condition

$$\begin{aligned} \rho &= \left[1 - \frac{S^2 Ma^2 (\gamma - 1) e^{2f}}{8\pi^2} \right]^{\frac{1}{\gamma-1}}, \\ u &= \frac{S(y - y_c) e^f}{2\pi R}, \\ v &= 1 - \frac{S(x - x_c) e^f}{2\pi R}, \\ p &= \frac{\rho^\gamma}{\gamma Ma^2}, \end{aligned} \quad (14.11)$$

where $f = (1 - (x - x_c)^2 - (y - y_c)^2)/2R^2$, $S = 13.5$ is the strength of the vortex, $R = 1.5$ is the characteristic vortex radius, $\gamma = 1.4$, the Mach number is $Ma = 0.4$, and x_c and y_c are the initial location of the vortex center. This initial condition is isentropic and, hence, without any viscous effects, the exact solution consists of the vortex maintaining its initial shape and simply advecting with the mean flow at a vertical velocity of unity. Since the domain is periodic, the vortex will eventually return to its initial condition, and at this point, we can evaluate an error norm of the density field of the form

$$\|\rho\|_2 = \sqrt{\frac{\sum_{i=1}^{N_x} \sum_{j=1}^{N_y} (\rho_{i,j} - \rho_{e,i,j})^2}{N_x N_y}}, \quad (14.12)$$

where $\rho_{e,i,j}$ is the exact solution at a grid point, which is equal to the initial condition at time $t = L_y/1$ since the vertical velocity is unity.

14.2 A Navier-Stokes Solver

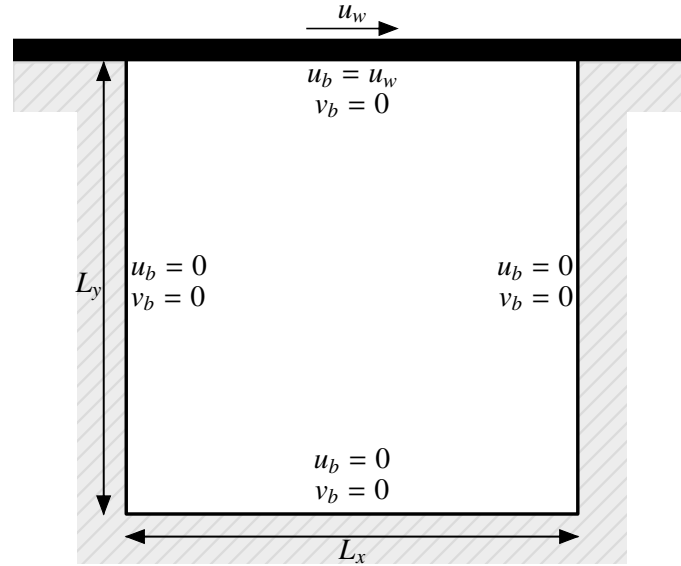


Figure 14.1: Lid-driven cavity flow problem

In order to solve the Navier-Stokes equations, we can readily extend the two-dimensional Euler solver outlined in the previous section. Here we will demonstrate how to solve the Navier-Stokes equations for a classical lid-driven cavity flow problem, as shown in Figure 14.1. Again, this will be discretized using the finite difference method in space, with grid spacings

$$\Delta x = \frac{L_x}{(N_x - 1)}, \quad (14.13)$$

and

$$\Delta y = \frac{L_y}{(N_y - 1)}, \quad (14.14)$$

to ensure that grid points are included along all edges of the domain.

Starting with the Navier-Stokes equations in two dimensions, we note that they can also be expanded in the following form

$$\frac{\partial \vec{w}}{\partial t} + \frac{\partial \vec{F}}{\partial x} + \frac{\partial \vec{G}}{\partial y} = 0, \quad (14.15)$$

where again

$$\vec{w} = [\rho, \rho u, \rho v, \rho E]^T, \quad (14.16)$$

is a vector of the conserved variables. In addition, unlike the isentropic vortex test case, the lid-driven cavity flow problem requires boundary conditions to be applied at the edges of the domain. In this case, we need to enforce the no-slip boundary conditions on all four walls. In addition, we will enforce that the walls are isothermal. This is achieved by setting

$$u_b = u_w, \quad v_b = v_w, \quad (14.17)$$

where $u_w = v_w = 0$ on all boundaries, except for the top plate where u_w is specified to be consistent with the chosen Reynolds number. Then, the temperature at the boundary is fixed at

$$T_b = T_w, \quad (14.18)$$

which is specified to be the same temperature as the initial gas in the domain. In order to get the density at the wall, the pressure is first projected from the interior onto the boundary, and

$$p_b = p_+, \quad (14.19)$$

where p_+ is the pressure at the first grid point off of the wall. Finally, the density can be obtained from the ideal gas law with the specified wall temperature and the pressure, and the state vector at the wall \vec{w}_b can be constructed.

For the Navier-Stokes equations, the flux functions can now be written as the sum of their respective inviscid and viscous components, such that $\vec{F} = \vec{F}_i + \vec{F}_v$ and $\vec{G} = \vec{G}_i + \vec{G}_v$, and \vec{F}_i and \vec{G}_i are the inviscid flux, which are still

$$\vec{F}_i = [\rho u, \rho u^2 + p, \rho uv, u(\rho E + p)]^T, \quad (14.20)$$

and

$$\vec{G}_i = [\rho v, \rho uv, \rho v^2 + p, v(\rho E + p)]^T. \quad (14.21)$$

Furthermore, the viscous fluxes \vec{F}_v and \vec{G}_v are expanded to

$$\vec{F}_v = [0, -\tau_{xx}, -\tau_{xy}, q_x - u\tau_{xx} - v\tau_{xy}]^T, \quad (14.22)$$

and

$$\vec{G}_v = [0, -\tau_{xy}, -\tau_{yy}, q_y - u\tau_{xy} - v\tau_{yy}]^T, \quad (14.23)$$

where

$$q_x = -k \frac{\partial T}{\partial x}, \quad (14.24)$$

$$q_y = -k \frac{\partial T}{\partial y}, \quad (14.25)$$

are the heat fluxes, k is the thermal conductivity and

$$\tau_{xx} = \frac{2}{3}\mu \left(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} \right), \quad (14.26)$$

$$\tau_{yy} = \frac{2}{3}\mu \left(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x} \right), \quad (14.27)$$

$$\tau_{xy} = \mu \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right). \quad (14.28)$$

We note that, unlike the Euler equations, we cannot obtain the viscous fluxes directly from the solution, since they also require the gradients of the velocity components and temperature. To obtain these gradients, we will simply use central differences in the middle of the domain. For example, for the gradient of u the following finite difference methods can be used

$$\frac{\partial u}{\partial x} = \frac{u_{i+1,j}^t - u_{i-1,j}^t}{2\Delta x}, \quad (14.29)$$

$$\frac{\partial u}{\partial y} = \frac{u_{i,j+1}^t - u_{i,j-1}^t}{2\Delta y}. \quad (14.30)$$

However, these finite difference methods will not work for computing the velocity and temperature gradients of the solution at the boundaries of the domain, since they would require points outside of the computational domain. Instead, one-sided stencils can be used at the wall. For example, the following stencil can be used on the left-hand side boundary of the domain

$$\frac{\partial u}{\partial x} = \frac{-u_{i,3} + 4u_{i,2} - 3u_{i,1}}{2\Delta x}, \quad (14.31)$$

and along the length of each wall, we will take the derivative to be zero. For example, on the left and right-hand side boundaries this will yield

$$\frac{\partial u}{\partial y} = 0. \quad (14.32)$$

Using these stencils we now have the solution vector at each grid point in the domain, and derivatives of the velocity and temperature field at each point in the domain. Now, at every point the inviscid and viscous fluxes can be computed. With these obtained, the same finite difference stencils can be applied to get the divergence of these fluxes, completing the right hand side of the conservation law. Finally, the solution can be advanced in time using a suitable explicit Runge-Kutta method for the temporal term.



Part 3: Applications

15	Introduction	137
16	Inviscid NACA 0012	139
17	Supersonic Wedge	149
18	Inviscid ONERA M6	161
19	Laminar Cylinder	170
20	Turbulent ONERA M6	183
21	Mesh Generation Using Gmsh	194
22	Shock Waves	205
	Bibliography	214
	Articles	
	Books	

15. Introduction

Gmsh



To read all about Gmsh and download its source, executables, and license please [click here](#).

Gmsh is a 2D/3D mesh generator including a built-in CAD engine and post-processor. Gmsh was developed to provide a fast, lightweight, and user-friendly meshing environment with additional visualization capabilities. It is open-source, and the source code and executables for various operating systems (Windows, Mac, Linux) are available for free. Gmsh contains modules for geometry definition, meshing, solving and post-processing. Each module has its own set of commands and utilities, which can be easily manipulated using a graphical user interface (GUI).

Upon launching the Gmsh executable, the GUI will open and the module panel will appear on the left-hand side of the window with headings to control the:

- **Geometry**
- **Mesh**
- **Solver**

The **Geometry** section is a module that allows you to create your own geometry directly within Gmsh or import external CAD files. Geometry in Gmsh is defined using a hierarchical structure. This means a volume is made up of different surfaces, and each surface is made up of different curves or lines. In turn, each curve or line is bounded by two end points in the case of a straight line, or a sequence of points in the case of a spline, for example. Therefore, to define a given geometry, a set of points is specified first, and the connections between these points are built using straight lines or curves. Then, a set of surfaces is created using the lines and curves, which are themselves used to define a volume in the case of a three-dimensional problem.

The geometry module has some key features to allow you to accomplish these tasks. Some of the important features are

- **Elementary entities:** Create, merge, split or delete points, lines, curves, faces or volumes.
- **Physical groups:** Specify boundary conditions and physical properties of a particular geometric entity.
- **Reload script:** Re-read the geometry saved in your .geo file, the Gmsh native file format.

- **Edit script:** Edit the .geo file manually using a built-in text editor.

It is worth noting that geometry specifications are saved in the .geo file in plain text format. This allows you to manually manipulate your geometry or any other specifications by editing this file using either the in-built text editor, or an external one of your choice.

After building the geometry and defining the boundary conditions via the **Geometry** module, the next step is usually to generate a mesh of the geometry using the **Mesh** module. This module splits the geometry up into a number of simple geometric elements, such as: lines, triangles, tetrahedra, hexahedra and pyramids. Gmsh has several different algorithms for automatically generating a mesh, and an unstructured mesh will be generated by default. The **Mesh** module has sections that help to specify the type, number, and density of the mesh in the computational domain. Some of the important sections are

- **Define:** Set the number of points and stretching ratios along lines. Additionally, control the mesh type (whether structured or unstructured).
- **2D/3D:** Generates the mesh for surfaces/volumes in the domain. It automatically generates the mesh using specifications provided in the **Define** section.

There is another module named **Solver**, which allows a numerical solver to be accessed directly from within Gmsh. In the context of the current book, this will not be used.

SU2



To read all about SU2 and download its source, executables, and license please [click here](#).

As discussed in the Physics and Numerics parts of this book, ultimately, an approximate solution to the Navier-Stokes or RANS equations is obtained by solving a discrete system of equations on a computational grid or mesh. SU2 is an open-source CFD solver built specifically for this task. SU2 is written in C++, and its primary applications are computational aerodynamics and shape optimization. SU2 is generally user-friendly, and pre-compiled versions are available for all major operating systems (Windows, Mac, Linux). In the following tutorials, we will demonstrate the utility of SU2 as a CFD tool for computational aerodynamics.

Paraview



To read all about Paraview and download its source, executables, and license please [click here](#).

After SU2 runs, it generates a set of data files, which includes the complete flow field solved on the mesh generated previously using Gmsh. This data must be post-processed to visualize flow structures of interest, such as the velocity or pressure fields. Paraview is an open-source and multi-platform visualization package designed for this task. It is also available for all operating systems (Windows, Mac, Linux) and is free to download and use. Paraview is designed specifically to handle large complex data sets, such as those generated in CFD simulations. In the following laboratory experiments, we will use Paraview as a post-processing tool for the analysis of the data produced by SU2. Typically, this analysis consists of contours of flow parameters, extracting line plots, slices, or pressure coefficient distributions.

16. Inviscid NACA 0012

Required Files



Use the following links to download the same version of SU2 for Windows ([click here](#)) or Mac ([click here](#)), the required configuration and mesh files ([click here](#)), and the reference dataset ([click here](#)).



Use the following links to download the same version of Paraview for Windows ([click here](#)) or Mac ([click here](#)).

Problem Description

In this tutorial, we are going to demonstrate how to simulate transonic inviscid flow around a NACA 0012 airfoil. Under the assumption of inviscid flow, the Euler equations will be used. Please note that this assumption is only reasonably valid at high Reynolds numbers and low angles of attack since the contribution of the viscous terms in the Navier-Stokes equations is minimal in this case. The flow specifications are provided as

- Pressure = 101,325 Pa
- Temperature = 273 K
- Mach number = 0.8
- Angle of attack = 1.25 degree

This tutorial has two parts: Flow Solution and Post-Processing. In the first part we will explain how to manage the required files and settings in SU2, and then run the simulation. In the second part we will demonstrate how to use Paraview to visualize the data files generated in the first step using SU2.

Flow solution

To run this simulation, SU2 needs two files: a configuration file (.cfg) and a mesh file (.su2). Links to the required files and executables are provided at the start of this tutorial. The files include:

1. inv_NACA_0012.cfg: the configuration file.
2. mesh_NACA_0012_inv.su2: the mesh file.

The next step is to copy these two files in the directory where you have saved the SU2 executable, so that everything is located in the same folder. Then, to run the simulation using the executable, mesh, and configuration files, simply open a terminal window and enter the following commands:

Windows	\$ cd "where you saved the package" \$ SU2_CFD.exe inv_NACA0012.cfg
Mac	\$ cd "where you saved the package" \$./SU2_CFD inv_NACA0012.cfg

The SU2 solver will commence solving the problem and will print out the residuals at every iteration until the specified convergence criteria is achieved. The computational time for this case is highly dependant on the computer's performance. However, the run time is expected to be about 15 minutes on average. After the calculations are complete, the following output files should have been generated within the SU2 folder:

- flow.vtk: The flow solution on the entire domain.
- force_breakdown.dat: Forces and moment on the airfoil.
- history.vtk: Convergence history.
- restart_flow.dat: Restart file.
- surface_flow.vtk: The flow solution on the airfoil surface.
- surface_flow.csv: A comma separated value file of the flow solution on the airfoil.

Please keep in mind that every time you run SU2, the output data will be overwritten. Hence, before launching a new simulation you should backup your files in another directory.

Post-Processing

In this section, we will explain how to use Paraview to visualize the solution files generated by SU2. First of all, install Paraview using the links at the start of this tutorial. Once that is complete, perform the following steps to visualize the results:

16.0.1 Load the Solution File

- 1) Launch Paraview.
- 2) Go to **File** → **Open**, and then select the flow.vtk file. On the left-hand side of the Paraview window, you will see the file appear under **builtin** in the **Pipeline Browser**.
- 3) Now press the **Apply** button in the **Properties** tab, right under the **Pipeline Browser** heading. After taking these steps, your file is loaded by Paraview and is ready to be visualized (Figure 16.1).

16.0.2 Visualize the Mesh

In order to view the mesh that was stored in the .su2 file, and as shown in Figure 16.2, select *Solid Color* with *Wireframe* in the toolbar. Then, you can zoom in to see the mesh near the surface of the airfoil, as shown in Figure 16.3. The mesh around the NACA 0012 is unstructured, and the elements are clustered around the leading and trailing edges to resolve the complex flow structures that are expected there.

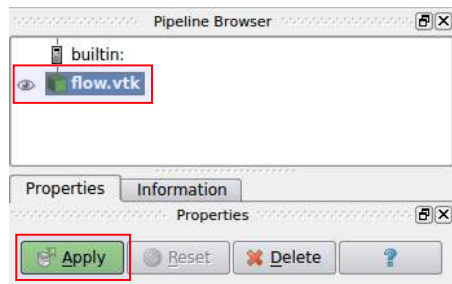


Figure 16.1: Loading the .vtk file into the **Pipeline Browser**.



Figure 16.2: Displaying the NACA 0012 mesh.

16.0.3 Visualize Pressure Contours

In order to visualize pressure contours, you can take the following steps:

- 1) Click on `flow.vtk` from within the **Pipeline Browser** to select the current data file. Then click on the **Properties** tab.
- 2) According to Figure 16.4, in the **Coloring** section select *Pressure* from the drop-down menu.

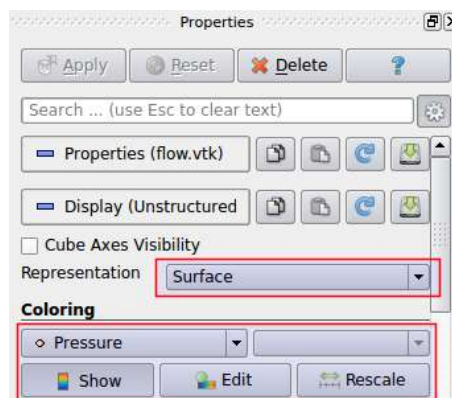


Figure 16.4: Contour settings in the **Display** tab.

- 3) To change the color settings used to show the pressure field, you can click on **Edit** under the **Coloring** options. Another display window appears on the right-hand side of the monitor, similar to Figure 16.5. Now you can change the maximum/minimum range of pressure to your desired values using the **Set Range** option, or change the contour colors using **Choose Preset** field (Figure 16.5). After these steps, the pressure contours shown in the display window should be similar to those shown in Figure 16.6.
- 4) To add contour lines, click again on the `flow.vtk` file in the **Pipeline Browser**, and then click on the **Contour** icon (Figure 16.7) in the toolbar. Now you should see that a new item called *Contour1* appears under `flow.vtk` in the **Pipeline Browser** (Figure 16.8).
- 5) Go to the **Properties** tab (as shown in Figure 16.9a), and select *Pressure* from the **Contour By** drop-down menu.
- 6) Click on the **New Range** icon to customize the range of contour lines that will be generated.
- 7) For now, set the number of steps to 20, similar to Figure 16.9b. This means the pressure

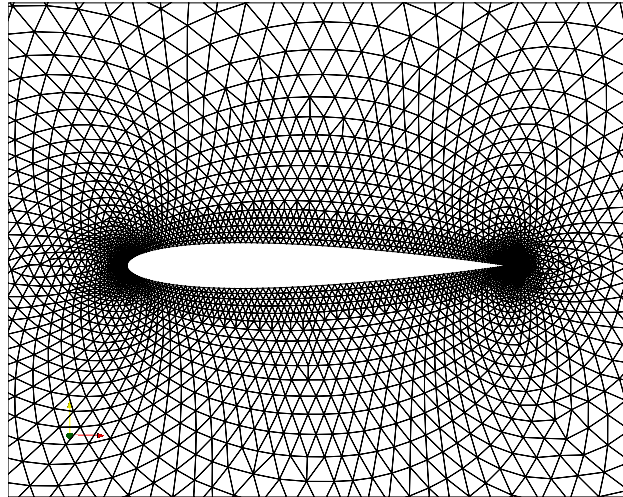


Figure 16.3: The unstructured mesh around the NACA 0012 airfoil.

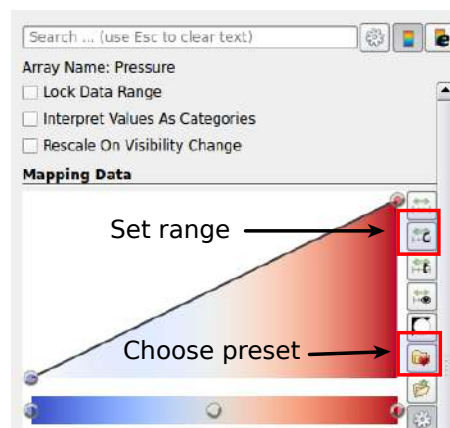


Figure 16.5: How to change color and max/min values for contours.

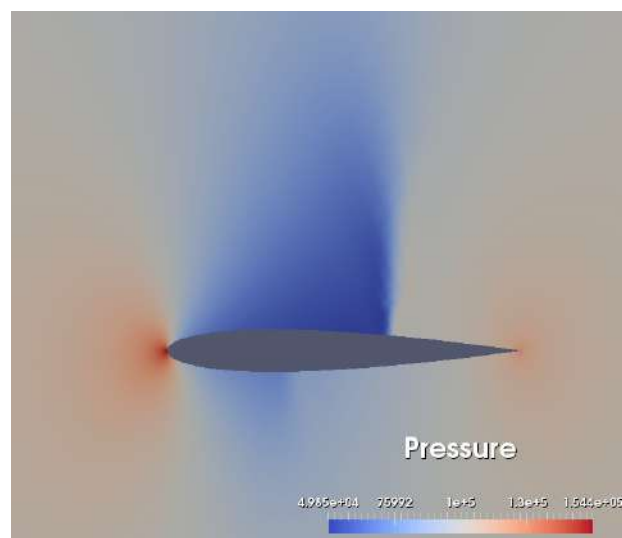
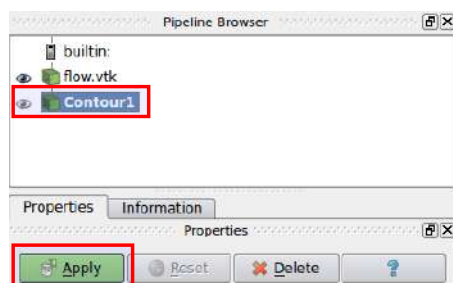


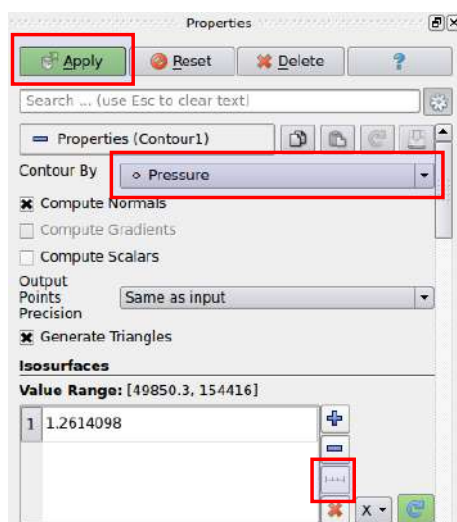
Figure 16.6: Pressure contour for NACA 0012 airfoil.



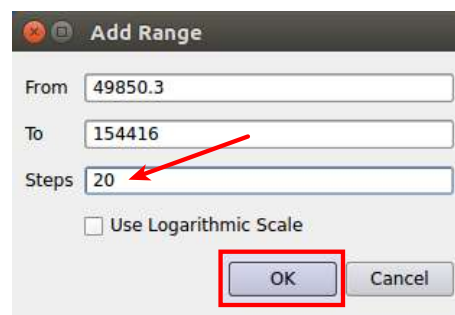
Figure 16.7: Contour icon in the toolbar.

Figure 16.8: Adding *Contour1* in **Pipeline Browser**.

contour range is equally divided by 20 portions between the minimum and maximum values. Finally, click on **Apply** to generate these contour lines in the display window.



(a) Define a new range



(b) Add range

Figure 16.9: How to define a new range for the contour lines.

- 8) As shown in Figure 16.10, click on **Display** under the **Properties** tab. In the **Coloring** section, select **Solid Color** from the drop-down menu, and choose white as the color using **Edit**. Now the pressure contour lines should look similar to those in Figure 16.11.

16.0.4 Visualize the Pressure Coefficient

The pressure data on the surface of the airfoil is stored in `surface_flow.vtk`. We will now generate a plot of the pressure coefficient as a function of chord-wise position. To do so, you can take the following steps:

- 1) Go to **Open** → **File** and select `surface_flow.vtk`. As shown in Figure 16.12, this file is now loaded and added to the list of items under **builtin** in the **Pipeline Browser**. Note that there is an eye icon on the left-hand side of each item in the **Pipeline Browser**, which

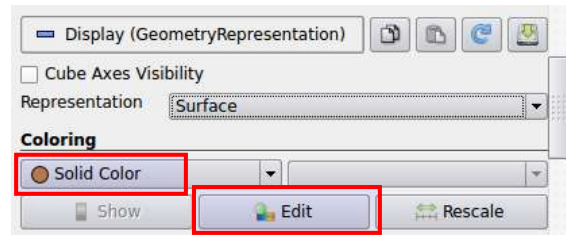


Figure 16.10: Changing contour line colors in the **Coloring** section.

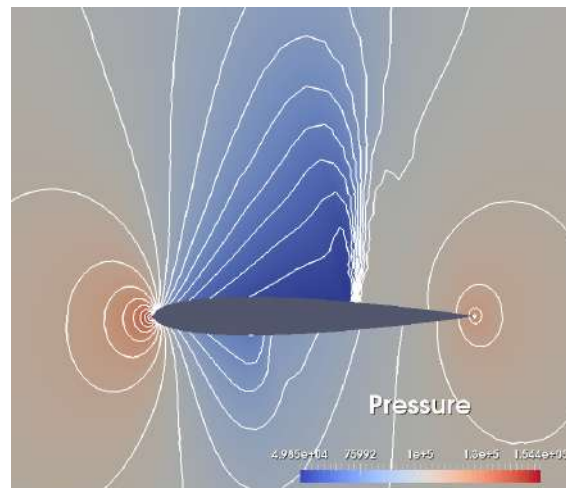


Figure 16.11: Pressure contours superimposed with contour lines around the NACA 0012 airfoil.

enables you to hide/unhide the plots related to each item. Since we want to see only the pressure coefficient plot, we hide the previously-generated contours by unselecting the eye icon beside `flow.vtk` and `Contour1`.



Figure 16.12: Loading the surface `.vtk` file into Paraview.

- 2) Select `surface_flow.vtk` in the **Pipeline Browser** (as shown in Figure 16.13). Then, go to **Filters** → **Search** (Figure 16.13a) and search for **Plot Data** (Figure 16.13b). After taking this step, as shown in Figure 16.14, the `PlotData1` item is added to the **Pipeline Browser**.
- 3) Hide (deactivate) all items in the list of the **Pipeline Browser** except `PlotData1` by clicking on the eye icons beside each, and then click on **Apply**.
- 4) As shown in Figure 16.15, under **Display** in the **Properties** tab, deactivate **Use Index For XAxis**, and then select `Points_X` from the drop-down menu.
- 5) Under **Series Parameters** in the same tab, unselect all variables except for `Pressure_Coefficient`. This allows you to have only one plot showing the pressure coefficient versus chord wise

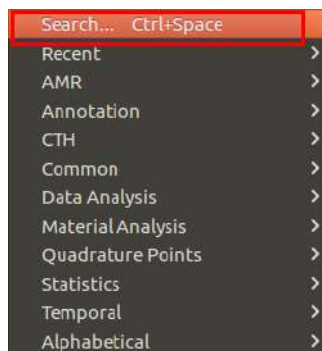
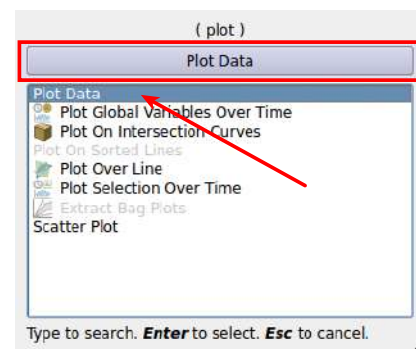
(a) Searching for a **Filter**(b) Searching for the **PlotData** filter

Figure 16.13: How to plot data.

position. The pressure plot in the display window should be similar to that shown in Figure 16.16.

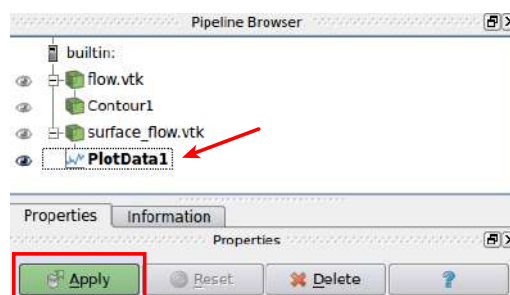
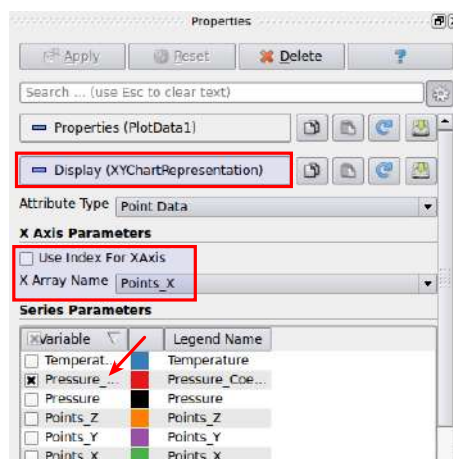
Figure 16.14: Adding *PlotData1* to the **Pipeline Browser**.

Figure 16.15: Plot settings for pressure coefficient along the airfoil surface.

By convention, $-C_p$ is typically plotted such that the pressure curve on suction side (lower pressure) is on top of the pressure curve on the pressure side (higher pressure). Therefore, we need to reverse the y-axis in this plot to flip it. To do this, you can take the following steps:

- 1) Go to **Display** in the **Properties** tab.
- 2) Find the **Left Axis Range** section in the same tab, and then click on **Left Axis Use Custom Range** (Figure 16.17).

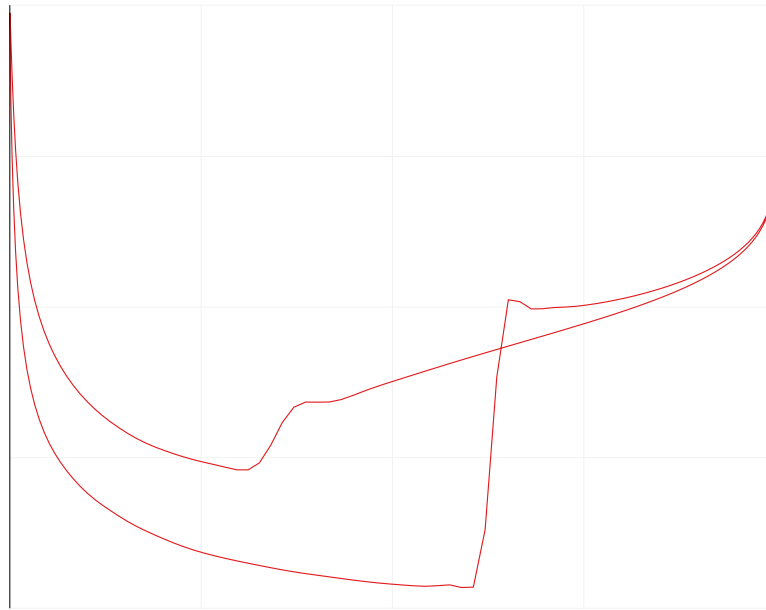


Figure 16.16: Pressure coefficient on the surface of the NACA 0012 airfoil.

- 3) Later, another section appears to select maximum/minimum ranges for the left axis of the plot. Now, switch numbers in these two text bars. This allows the y-axis to be reverse (flipped).
- 4) There are other parameters that you may want to change in the same tab, such as the **Left Axis Title** or the **Bottom Axis Title**. Please type C_p and x/c in **Left Axis Title** and **Bottom Axis Title**, respectively.
- 5) For adding markers to the plot lines, from **Display** select the **Marker Style** as *Diamond* (Figure 16.18). You can also change the thickness of the plot line in the same tab.

After taking all these steps, the final plot you get should be similar to Figure 16.19.

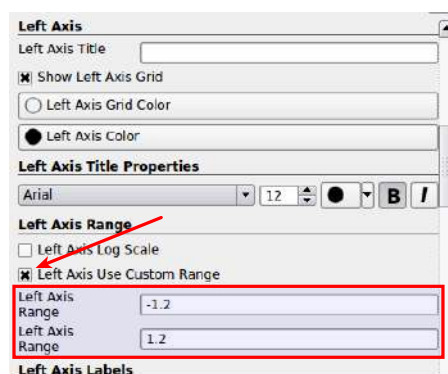


Figure 16.17: Plot Settings.

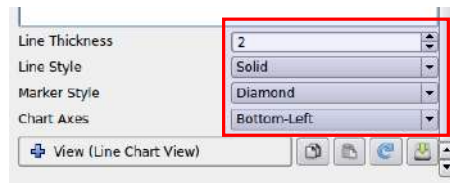


Figure 16.18: How to add markers to a line plot.

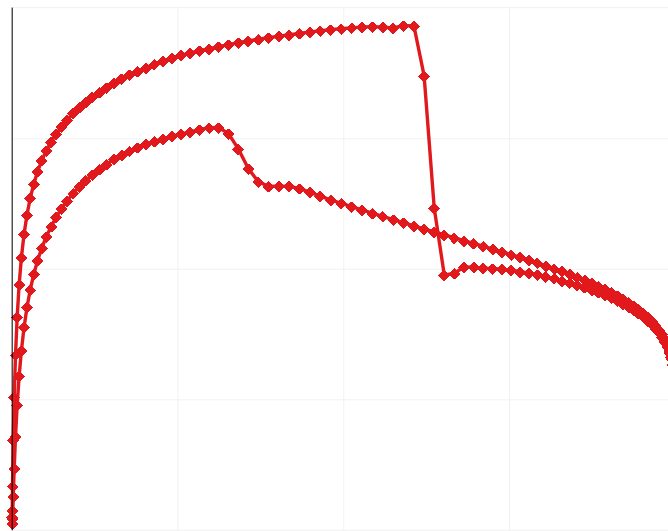


Figure 16.19: Final pressure coefficient plot on the surface of the NACA 0012 airfoil.

16.0.5 Aerodynamic Forces

In order to obtain the aerodynamic forces on the airfoil, open `force_breakdown.dat` using a text editor. As shown in Figure 16.20, the flow properties are tabulated and it can be confirmed that they agree with the configuration file.

```
Problem definition:

Inviscid flow: Computing density based on free-stream
temperature and pressure using the ideal gas law.
Force coefficients computed using free-stream values.
-- Input conditions:
Fluid Model: STANDARD_AIR
Specific gas constant: 287.058 N.m/kg.K.
Specific gas constant (non-dim): 287.058
Specific Heat Ratio: 1.4000
Free-stream static pressure: 101325 Pa.
Free-stream total pressure: 154454 Pa.
Free-stream temperature: 273.15 K.
Free-stream total temperature: 308.113 K.
Free-stream density: 1.29225 kg/m^3.
Free-stream velocity: (264.994, 5.78219) m/s. Magnitude: 265.057 m/s.
Free-stream total energy per unit mass: 231152 m^2/s^2.
```

Figure 16.20: Fluid and flow properties in `force_breakdown.dat`.

As shown in Figure 16.21, the aerodynamic forces are expressed in non-dimensional form by using free-stream values for density and velocity, as well as a unit reference length in this case. The actual dimensional forces can be obtained by multiplying the flow coefficients with these

non-dimensional factors calculated with the free-stream density and velocity. Note that you can find the lift coefficient (C_L), drag coefficient (C_D), lift to drag ratio (C_L/C_D), the moment coefficient ($C_{M,z}$), x -component of force coefficient ($C_{F,x}$) and y -component of force coefficient ($C_{F,y}$) all from the `force_breakdown.dat` file.

```
Forces breakdown:
Center of Pressure: X=0.043421Y+0.352950 m.
NOTE: Multiply forces by the non-dimensional
factor: 45393.600000, and the reference factor: 1.000000
to obtain the dimensional force.
Total CL:      0.326934 | Pressure ( 99%): 0.326934
Total CD:      0.021350 | Pressure ( 99%): 0.021350
Total CL/CD:   15.313238 | Pressure (100%): 15.313238
Total CMz:     0.033698 | Pressure ( 99%): 0.033698
Total CFx:     0.014213 | Pressure ( 99%): 0.014213
Total CFy:     0.327322 | Pressure ( 99%): 0.327322
```

Figure 16.21: Aerodynamic forces obtained from `force_breakdown.dat`.

Questions

- Run the provided default NACA 0012 test case at $Ma = 0.8$.
 - Plot and comment on the mesh.
 - Plot and comment on the pressure contours around the airfoil.
 - Plot and comment on the pressure coefficient on the surface of the airfoil.
- Re-run the default NACA 0012 case but change the Mach number to $Ma = 0.3$, and run several simulations using $\alpha = 0, 2, 4, 6, 8, 10, 12, 14, 16$ degrees.
 - Plot C_L vs α alongside the provided experimental data provided at the start of this tutorial [10].
 - Repeat 1.b with $Ma = 0.3$ for the $\alpha = 0, 8, 16$ degree cases.
 - Repeat 1.c with for the $\alpha = 0, 8, 16$ degree cases, and include the provided experimental data from [11].
- Compare your CFD results in Q.2, and discuss sources of error that could have led to any discrepancies in your results relative to the experimental data.

17. Supersonic Wedge

Required Files



Use the following links to download the same version of SU2 for Windows ([click here](#)) or Mac ([click here](#)), and the required configuration and mesh files ([click here](#)).



Use the following links to download the same version of Paraview for Windows ([click here](#)) or Mac ([click here](#)).

Problem Description

In this tutorial, we are going to demonstrate how to simulate supersonic inviscid flow past a simple 2D wedge. This wedge generates an oblique-shock wave moving outwards from its surface. Assuming the Reynolds number is high, and that the fluid is an ideal gas, we will be using the Euler equations. The domain is approximated using a structured mesh with 3,750 nodes. The domain consists of a flat upper wall, and a lower wall with an inclined a wedge starting at $x/L = 1/3$, where L is the length of the duct. The wedge angle is taken to be 10 degrees, and the inlet flow parameters are

- Pressure = 101,325 Pa
- Temperature = 273.15 K
- Mach number = 2.0
- Wedge angle = 10 degrees

This tutorial has two parts: Flow Solution and Post-processing. In the first part, we will explain how to manage the prerequisite files and settings, and how to run the CFD simulation using SU2. In the second part, we explain how to use Paraview to visualize the data obtained from SU2.

Flow solution

To run this simulation, SU2 needs two files: a configuration file (.cfg) and a mesh file (.su2). Links to the required files and executables are provided at the start of this tutorial. The files include:

1. `inv_wedge_HLLC.cfg` which is a configuration file.
2. `mesh_wedge_inv.su2` which is a mesh file.

The next step is to copy these two files into the same directory as the SU2 executable. Then, to run the simulation, open a terminal window and enter the following commands:

Windows	\$ cd "where you saved the package" \$ SU2_CFD.exe inv_wedge_HLLC.cfg
Mac	\$ cd "where you saved the package" \$./SU2_CFD inv_wedge_HLLC.cfg

The SU2 solver will commence solving the problem and will print out the residuals at every iteration, until the specified convergence criteria is achieved. The computational time for this case is highly dependant on the computer's performance. However, the run time is expected to be about 15 minutes on average. After the calculations are complete the following output files should have been generated within in the SU2 folder:

- `flow.vtk`: The flow solution on the entire domain.
- `history.vtk`: Convergence history.
- `restart_flow.dat`: Restart file.
- `surface_flow.vtk`: The flow solution on the airfoil surface.

Please keep in mind that every time you run SU2, the output data will be overwritten. Hence, before launching a new simulation you should backup your files in another directory.

Post-Processing

In this section, we explain how to use Paraview to visualize the solution generated by SU2. First of all, install paraview (if not already done) using the links at the start of this tutorial. Once that is complete, perform the following steps to visualize the results:

17.0.1 Load the Solution File:

- 1) Launch Paraview.
- 2) Go to **File** → **Open**, and then select the `flow.vtk` file. On the left side of the Paraview window, you will see the file appears in the **Pipeline Browser** under **builtin**
- 3) Now press the **Apply** button in the **Properties** tab, right under the **Pipeline Browser** heading. After taking these steps, your file is loaded by Paraview and is ready to be visualized (Figure 17.1).

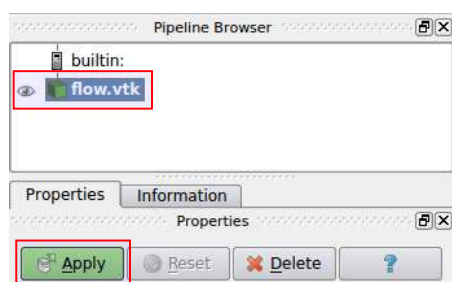


Figure 17.1: Loading the .vtk file into the **Pipeline Browser**.

17.0.2 Visualize the Mesh

As shown in Figure 17.2, in order to view the mesh select *Solid Color* with *Wireframe* in the toolbar. Then, you can zoom in to see the mesh for the computational domain, as shown in Figure 17.3. As you can see, the mesh in the duct is structured, and the wedge abruptly changes the direction of mesh on the bottom surface. Additionally, the grids are approximately uniform throughout the domain.

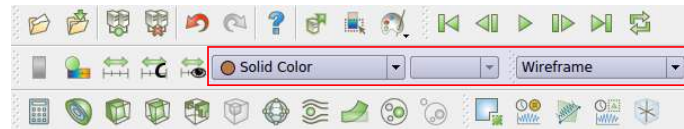


Figure 17.2: How to display the mesh.

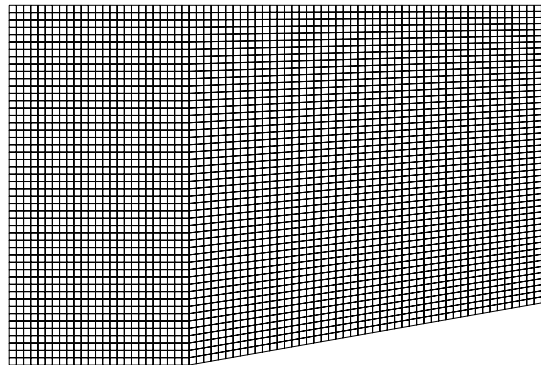


Figure 17.3: The structured mesh for the supersonic wedge.

17.0.3 Visualize Pressure and Mach Contours

To display pressure contours, you can take the following steps:

- 1) Click on `flow.vtk` in the **Pipeline Browser**, and then click on the **Display** form in the **Properties** tab.
- 2) Under the **Coloring** section select *Pressure* from the drop-down menu (Figure 17.4).

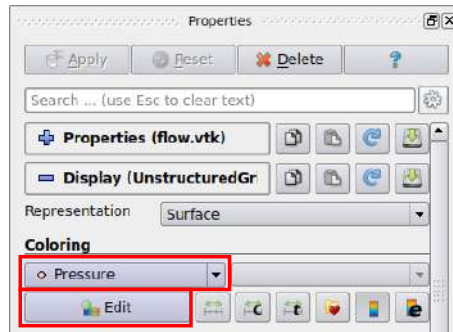


Figure 17.4: Settings for displaying pressure contours.

- 3) Additionally, to change the color settings used for pressure, you can click on the **Edit** option under the **Coloring** section. Another display window will appear on the right-hand side of the menu entitled **Mapping Data**, similar to Figure 17.5. Now you can change contour colors by choosing **Choose preset**.

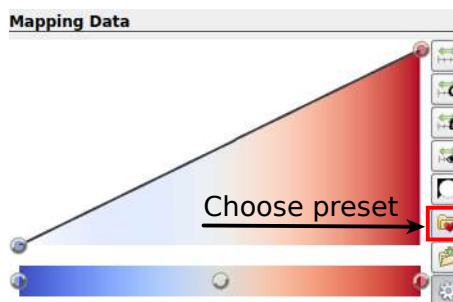


Figure 17.5: Changing the color range used for pressure contours.

- 4) Choose a color scale for your contours. Here, we select the *Cold and Hot* color scale to show clearly the sharp changes in pressure values around the shock (Figure 17.6). Then, click **Apply**.

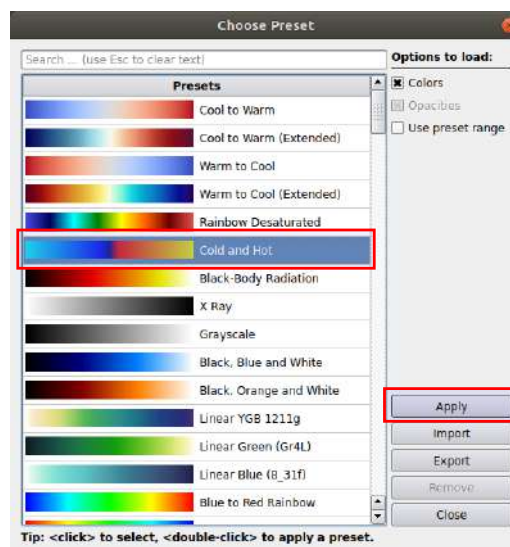


Figure 17.6: Different color ranges used to display contours.

- 5) To add axes to the plots, look under the **Miscellaneous** form of the **Display** field (Figure 17.7) and select the check box beside **Data Axes Grid**. Then, go to **Edit** and change these options based on your preferences (Figure 17.8). Finally, the pressure contour should be similar to Figure 17.9.

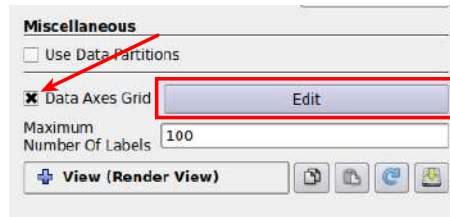


Figure 17.7: How to add axes to the plots.

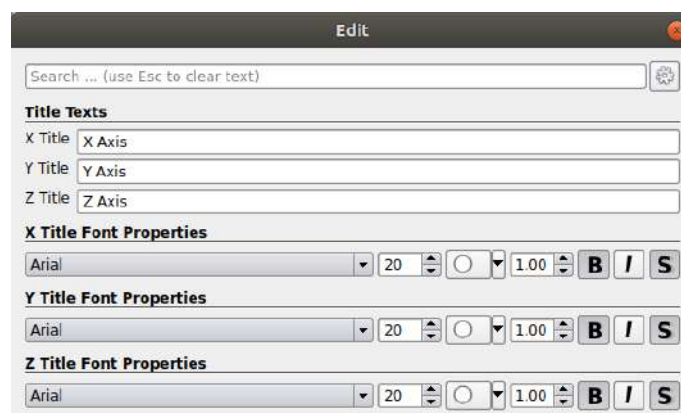


Figure 17.8: Settings for the axis style.

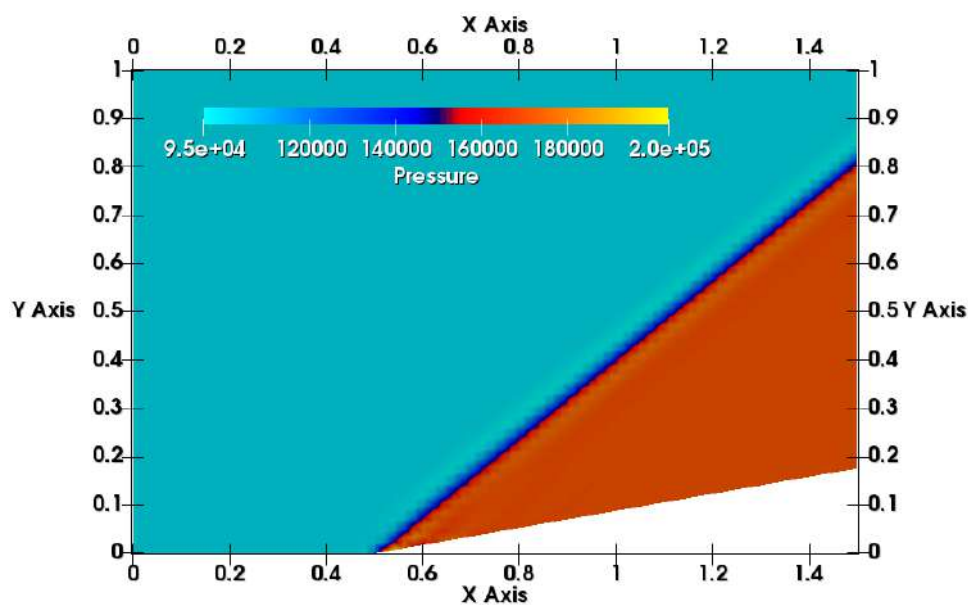


Figure 17.9: Pressure contours for the supersonic wedge case.

The pressure gradient along the shock wave is difficult to visualize directly from these contours.

To help, we can add contour lines to clearly delineate the different regions. To add contour lines, you can take the following steps:

- 6) Click on the `flow.vtk` file in the **Pipeline Browser**, and then click on the **Contour** icon (Figure 17.10) in the toolbar. Now, a new *Contour1* item appears under the `flow.vtk` file in the **Pipeline Browser** (Figure 17.11). Click on **Apply** to proceed to the next step.



Figure 17.10: **Contour** icon in the toolbar.

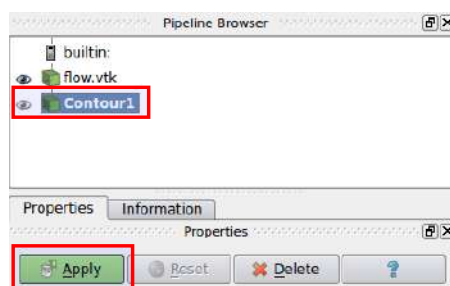
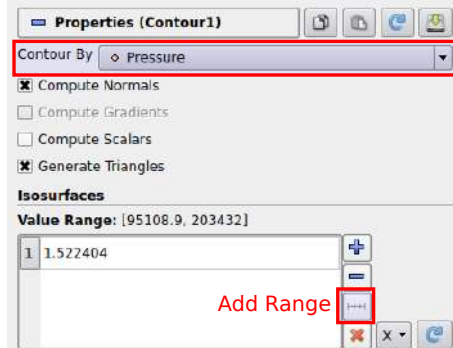
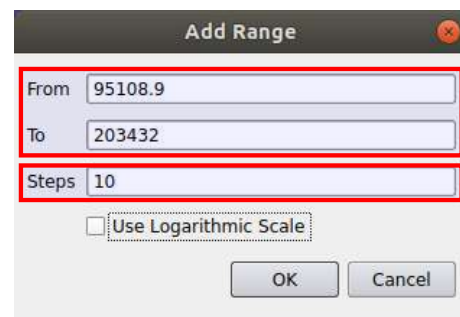


Figure 17.11: Adding *Contour1* to the **Pipeline Browser**.



(a) Define a new range



(b) Add range

Figure 17.12: How to define a new range for the contour lines.

- 7) Following Figure 17.12a, go to the **Properties** tab and select *Pressure* from the **Contour By** drop-down menu.
- 8) Click on the **Add Range** icon to customize the range for the pressure contours. In Figure 17.12b, you will see the max/min range of contour lines (i.e. **From/To**), as well as the number of lines you would like to have in your contour plot (i.e. **Steps**).
- 9) Next, set **Steps** to 20 and click **OK**. By doing this the pressure contour range is equally divided into 20 lines. At the end, click on **Apply** to see contour lines in the display window.
- 10) As shown in Figure 17.13, click on **Display** under the **Properties** tab. In the **Coloring** section select **Solid Color** from the drop-down menu and choose white from **Edit**. The pressure contour lines should now be similar to those in Figure 17.14. You can zoom-in to the region at the bottom of the wedge to see the pressure gradient along the oblique shock, as

shown in Figure 17.15.

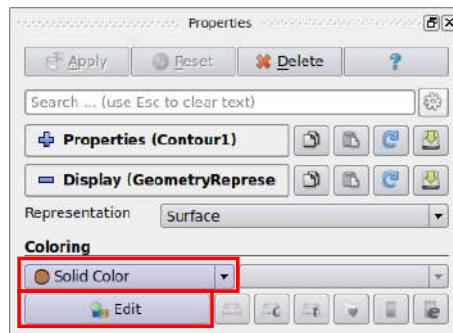


Figure 17.13: Changing contour line colors in the **Coloring** section.

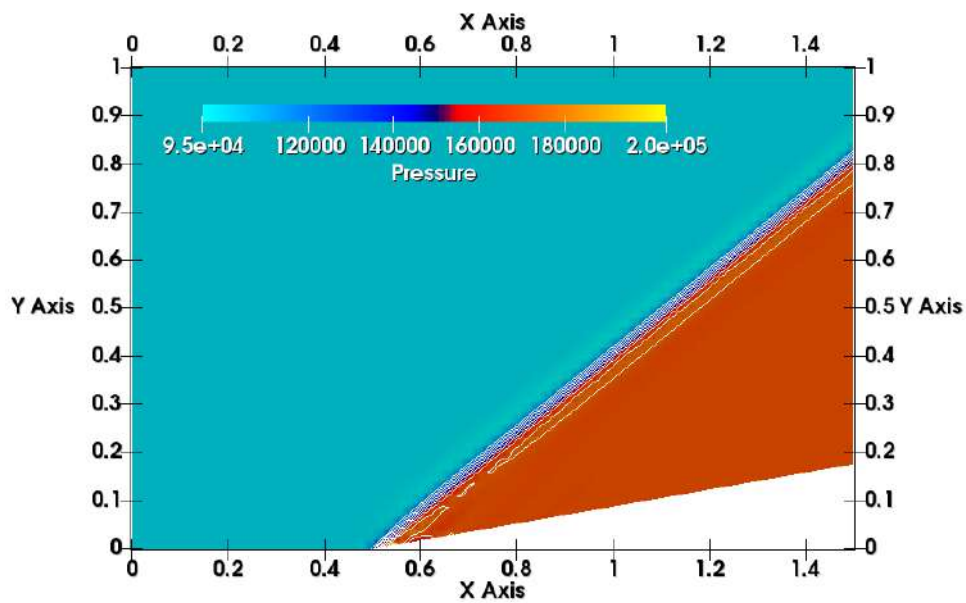


Figure 17.14: Pressure contours superimposed by contour lines for the supersonic wedge case.

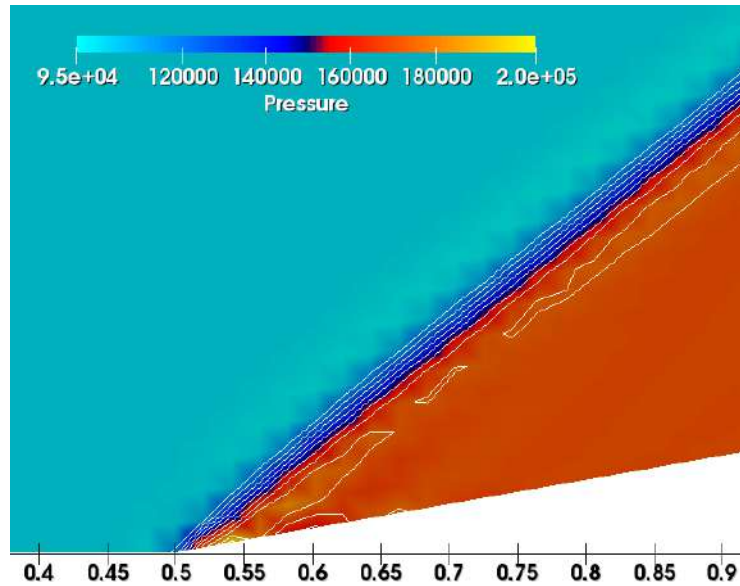
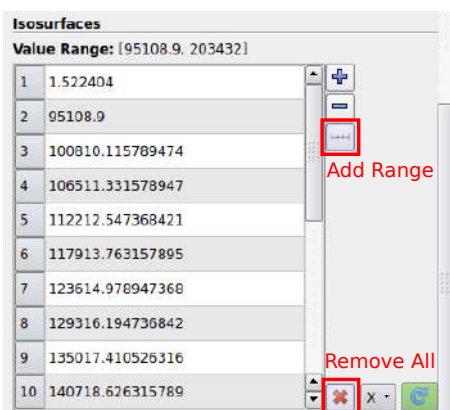
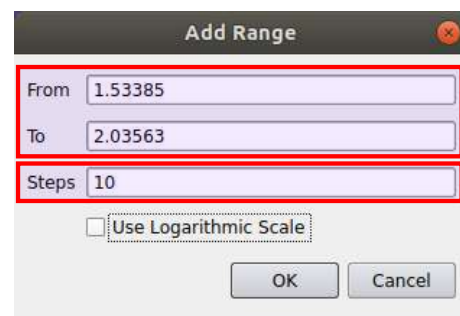


Figure 17.15: Magnified pressure contour superimposed with contour lines for the supersonic wedge case.

- 10) Now to display the Mach number contours click on `flow.vtk` in the **Pipeline Browser**.
- 11) Similar to Figure 17.4, under the **Coloring** section select *Mach* from the drop-down menu.
- 12) Click on *Contour1* in the **Pipeline Browser**. Now, under **Properties** select *Mach* from the **Contour By** drop-down menu.
- 13) Since these settings are related to the previous pressure values, we need to revise some options to display contours of the *Mach* number appropriately. To do this, we need to delete the data range that was used for pressure, and replace it with the data range corresponding to the Mach number. Under **Isosurfaces** from **Properties**, click on the **Remove All** icon and then click on the **Add Range** icon, as shown in Figure 17.16a. As you can see from Figure 17.16b, the max/min values for the Mach number have changed.
- 14) Set the **Steps** to 20 and click on **OK** to proceed to the next step. Now, the Mach number contours should look similar to Figure 17.17. Additionally, if you zoom-in the plot, you will see more contour details near the wedge, similar to Figure 17.18.



(a) Define a new contour range



(b) Add range

Figure 17.16: Define a new range for the contour lines.

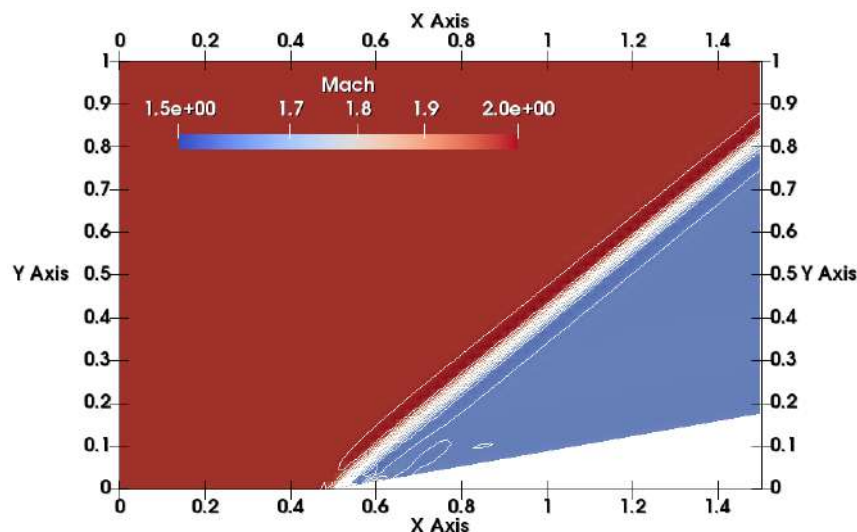


Figure 17.17: Mach number contours superimposed with contour lines for the supersonic wedge case.

17.0.4 Plotting a variable over an arbitrary line

To plot a variable over an arbitrary line in space, you can take the following steps:

- 1) Select `flow.vtk` by clicking on it in the **Pipeline Browser**.
- 2) Go to **Filters** → **Data Analysis** → **Plot Over Line** (as shown in Figure 17.19).
- 3) Under **Line Parameters** in **Properties** (as shown in Figure 17.20), select the coordinates for two arbitrary points you want (i.e. **Point1** and **Point2**). In this case, plot the line from **Point1** to **Point2** with (0,0.5,0) and (1.5,0.5,0), respectively, and then click **Apply**. A new line plot item will be generated.

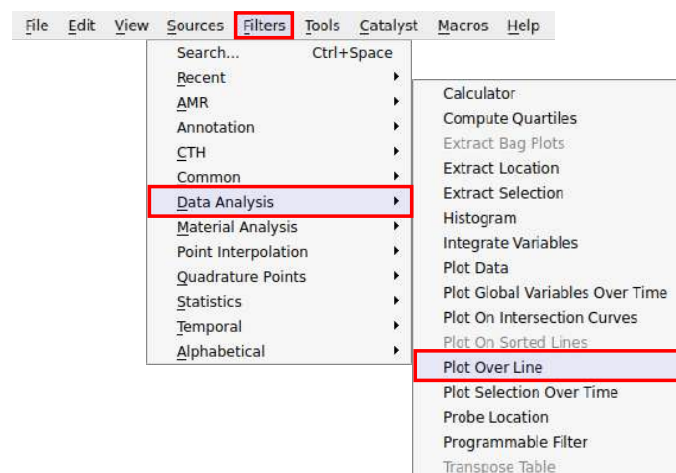


Figure 17.19: How to plot data over an arbitrary line in space.

- 4) According to Figure 17.21, under the **X Axis Parameters** from **Display**, select *Point_X* from the **X Array Name** drop-down menu. Now, under the **Series Parameters** options, toggle the box beside **Variable** to uncheck everything, then select only the *Mach* item. The plot of Mach number vs the *x-coordinate* will be displayed in the main window as shown in Figure 17.22.

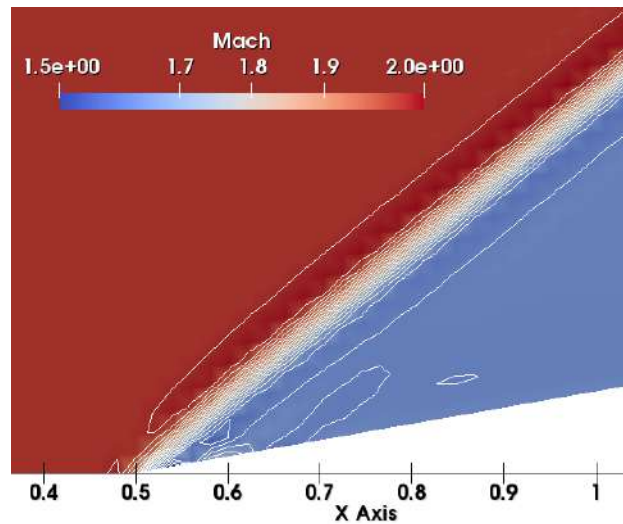


Figure 17.18: Magnified Mach number contours superimposed with contour lines for the supersonic wedge case.

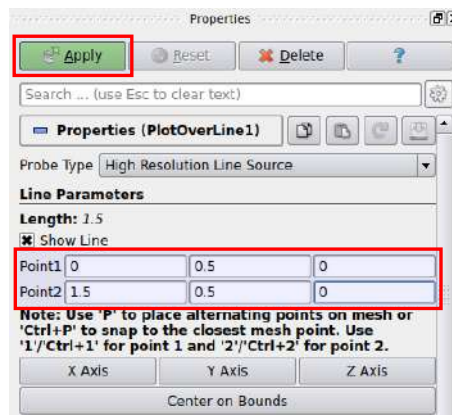


Figure 17.20: Define the coordinates for a line plot in space.

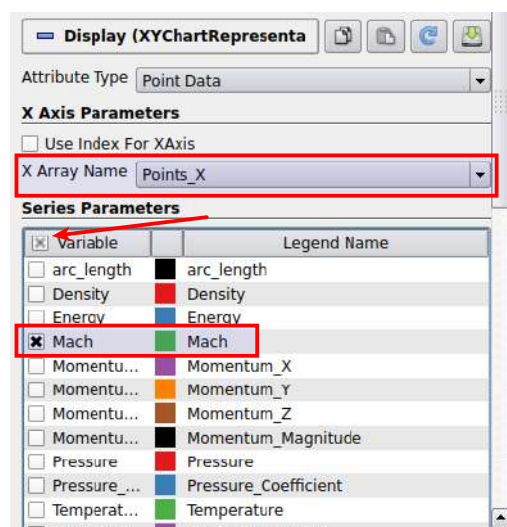


Figure 17.21: How to plot Mach number over a line.

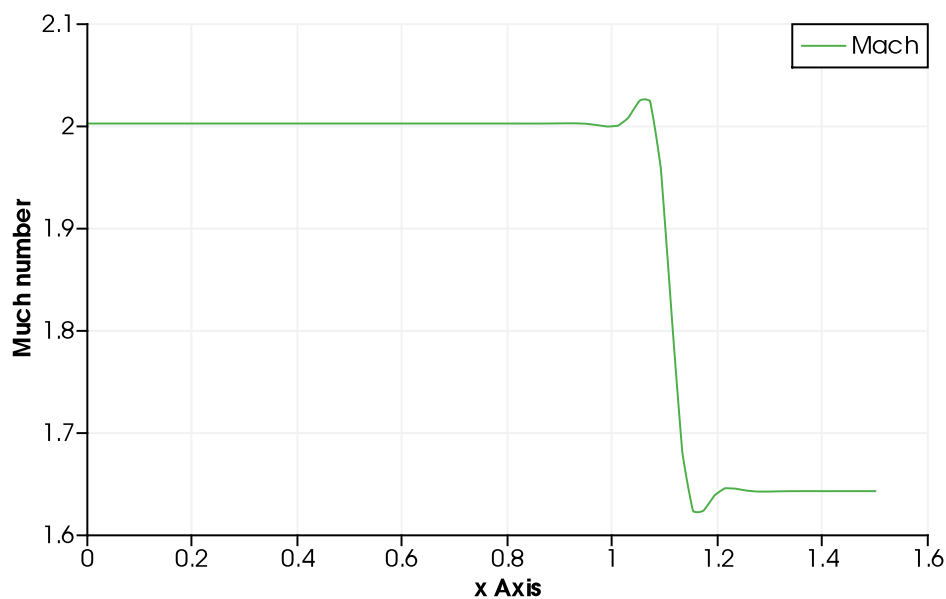


Figure 17.22: Mach number plotted over the specified line.

- 6) In order to see a spreadsheet view of the datapoints, you can click the upper right icon as shown in Figure 17.23, then click on the *Spreadsheet View* similar to Figure 17.24.
- 7) Additionally, you can export a .csv file and plot it with any other plotting software. To export the spreadsheet as a .csv file, go to **File** → **Export**, and then **Save as .csv**.



Figure 17.23: How to make spreadsheet view in a new window.

Questions

1. Run the default case as provided which uses 2ND_ORDER and the HLLC Riemann solver.
 - (a) Create coloured contours of Pressure and Mach number in the entire domain.
 - (b) Plot Pressure from (0,0.5,0) to (1.5,0.5,0) using **Plot Over Line**.
2. Repeat Q.1 but switch the SPATIAL_ORDER_FLOW to 1ST_ORDER.
3. Repeat 1 using SPATIAL_ORDER_FLOW as 2ND_ORDER but using the JST flux.
4. Repeat Q.1 using SPATIAL_ORDER_FLOW as 2ND_ORDER but using the LAX-FRIEDRICH flux.
5. Repeat Q.1 using SPATIAL_ORDER_FLOW as 2ND_ORDER but using the CUSP flux.
6. Repeat Q.1 using SPATIAL_ORDER_FLOW as 2ND_ORDER but using the ROE flux.
7. Comment on how the spatial order of accuracy and choice of Riemann solver affect the resolution of the shock and any dissipation or dispersion errors you can observe. Based on your results, recommend a combination of spatial order and Riemann solver that performs particularly well.

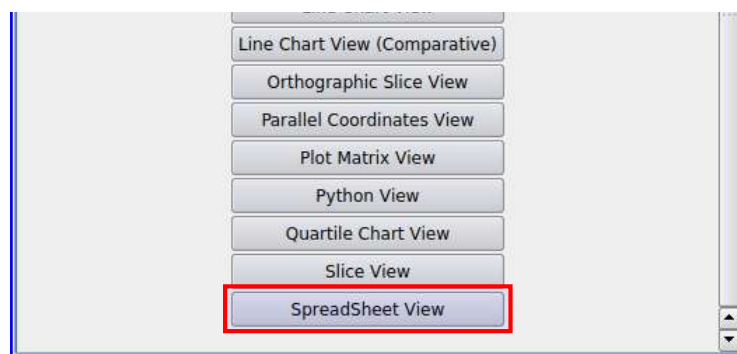


Figure 17.24: Selecting the *Spreadsheet View* among the different available views.

18. Inviscid ONERA M6

Required Files



Use the following links to download the same version of SU2 for Windows ([click here](#)) or Mac ([click here](#)), and the required configuration and mesh files ([click here](#)).



Use the following links to download the same version of Paraview for Windows ([click here](#)) or Mac ([click here](#)).

Problem Description

In this tutorial, we are going to explain how to simulate transonic inviscid flow around the three-dimensional ONERA M6 wing. This is a commonly used benchmark problem for computational aerodynamics due to the availability of experimental data and its relatively simple geometry. In this tutorial, the computational domain is discretized using an unstructured mesh with of 582,752 tetrahedral elements. The flow specification is as follows:

- Pressure = 101,325 Pa
- Temperature = 273.15 K
- Mach number = 0.8395
- angle of attack = 3.06 degrees

This tutorial has two parts: Flow Solution and Post-processing. In the first part, we will explain how to manage the prerequisite files and settings and how to run the CFD simulation using SU2. In the second part, we explain how to use Paraview to visualize the data obtained from SU2.

Flow solution

In the configuration file, you can manually adjust the multigrid control parameters. For the tutorial case, you will see that the number of multigrid levels is set to 0, `MGLEVEL=0`. In the assigned questions, the number of multigrid levels will be changed by adjusting this `MGLEVEL` parameter.

You can also select the multigrid cycle type by choosing one of either `MGCYCLE` (`V Cycle`, `W Cycle` or `Full MG Cycle`). Another parameter that can be controlled is the total number of iterations. This can be modified by changing the value of `EXT_ITER`. For this example, it will be kept at 200 for demonstration purposes, but depending on the method used, more iterations may be required to reach convergence. We will post-process the results to obtain the pressure coefficient and Mach number contours, as well as the lift coefficient (C_L), and drag coefficient (C_D) of the ONERA M6 wing. We will also explore the rate of convergence as a function of the number of multigrid levels used.

To run the simulation, SU2 needs two files: the configuration file (`.cfg`) and the mesh file (`.su2`). Links to the required files and executables are provided at the start of this tutorial. The files include:

1. `inv_ONERAM6_JST.cfg` as a configuration file.
2. `mesh_ONERAM6_inv.su2` as a mesh file.

The next step is to copy these two files into the same directory as your SU2 executable. Then, open a terminal window and execute the following commands to run the simulation:

Windows	<pre>\$ cd "where you saved the package" \$ SU2_CFD.exe inv_ONERAM6_JST.cfg</pre>
Mac	<pre>\$ cd "where you saved the package" \$./SU2_CFD inv_ONERAM6_JST.cfg</pre>

The solver will begin solving the problem and will print out the residuals at every iteration, until the specified convergence criteria is achieved. The computational time for this case is highly dependant on the computer's performance. However, the run time is expected to be about 5 hours on average. After the calculations are complete the following output files will be generated within in the SU2 folder:

- `flow.vtk`: The flow solution on the entire domain.
- `force_breakdown.dat`: Forces and moment on the airfoil.
- `history.vtk`: Convergence history.
- `restart_flow.dat`: Restart file.
- `surface_flow.vtk`: The flow solution on the airfoil surface.
- `surface_flow.csv`: A comma separated value file of the flow solution on the airfoil.

Please keep in mind that every time you run SU2, the output data will be overwritten. Hence, before launching a new simulation you should backup your files in another directory.

Post-processing

In this section, we will explain how to use Paraview to visualize the results produced by SU2.

18.0.1 Load the Solution File:

In order to import the data to Paraview, you can take the following steps:

- 1) Launch Paraview.
- 2) Go to **File** → **Open**, and select the `surface_flow.vtk` file.
- 3) On the left-hand side of the Paraview window you will see the file is loaded under **builtin** in **Pipeline Browser**. Press the **Apply** button in the **Properties** tab, right under the **Pipeline Browser** heading. Paraview will now load the data associated with your file, and it will be ready for visualization (Figure 18.1).

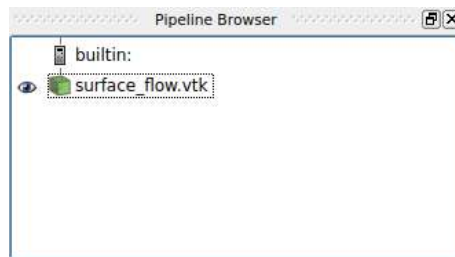


Figure 18.1: Loading `surface_flow.vtk` file into the **Pipeline Browser**.

18.0.2 Visualize the Mesh

In order to view the mesh, as shown in Figure 18.2, select *Solid Color* with *Wireframe* in the toolbar. As you can see in Figure 18.3, the mesh around the ONERA M6 wing is unstructured and the elements are clustered around the wingtip, leading edge, and trailing edge of the wing. This is to accommodate large changes in the flow solution in the vicinity of the features.



Figure 18.2: How to display the ONERA M6 mesh.

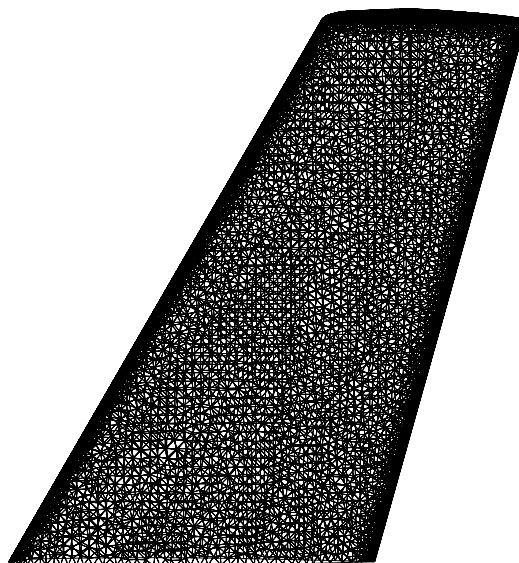


Figure 18.3: The unstructured mesh around the ONERA M6 wing.

18.0.3 Pressure Coefficient and Mach Number Contours

To display pressure coefficient contours on the surface of the wing, you can take the following steps:

- 1) Click on `surface_flow.vtk` in the **Pipeline Browser**.
- 2) Click on **Display** in **Properties** tab

- 3) Under **Coloring**, select *Pressure_Coefficient* from the drop-down menu (Figure 18.4)
- 4) Pressure coefficient contours should now be displayed, similar to Figure 18.5.

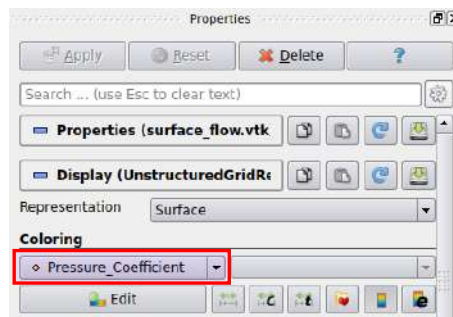


Figure 18.4: Display contours by selecting the appropriate variable.

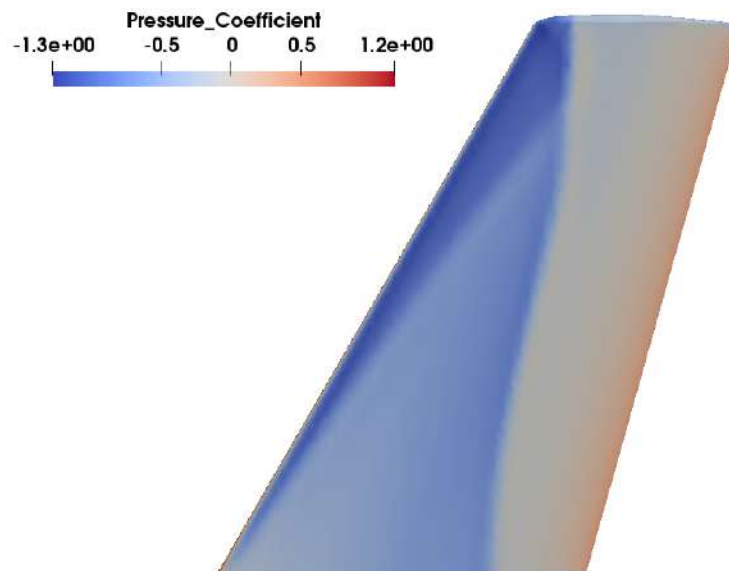


Figure 18.5: Pressure coefficient contour for ONERA M6 wing.

- 5) To add contour lines to the plot click on *surface_flow.vtk* in the **Pipeline Browser**, and then click on the **Contour** icon (Figure 18.6) in the toolbar.
- 6) The *Contour1* item now appears under *flow.vtk* file in the **Pipeline Browser** (Figure 18.7). Click on **Apply** to proceed to the next step.



Figure 18.6: **Contour** icon in toolbar.

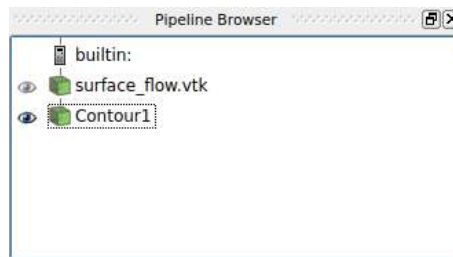
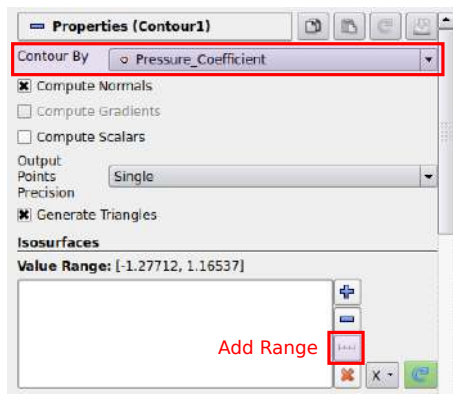
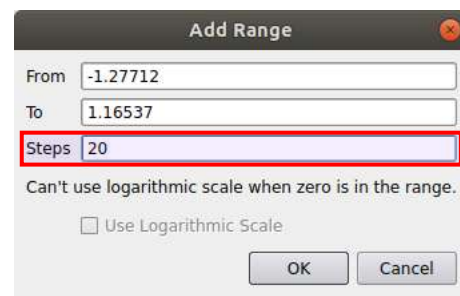


Figure 18.7: Adding *Contour1* in the **Pipeline Browser**.

- 7) As shown in Figure 18.8a, go to **Properties** and choose *Pressure_Coefficient* from **Contour By** drop-down menu
- 8) Click on the **Add Range** icon to customize the range used for the pressure contours.
- 9) As shown in Figure 18.8b, set the max/min (i.e. **From/To**) range of contour lines, as well as the number of steps used to divide this range into equally spaced contour levels. Set **Step** to 20, and then click on **OK**.
- 10) At the end, click on **Apply** to generate the contour lines in the display window.



(a) Define a new range



(b) Add the new range

Figure 18.8: How to define a new range for the contour lines.

- 11) As shown in Figure 18.9, click on **Display** under the **Properties** tab.
- 12) In the **Coloring** section select **Solid Color** from the drop-down menu and choose black from **Edit**.
- 13) Pressure coefficient contours will now be displayed in black and should be similar to those shown in Figure 18.10.

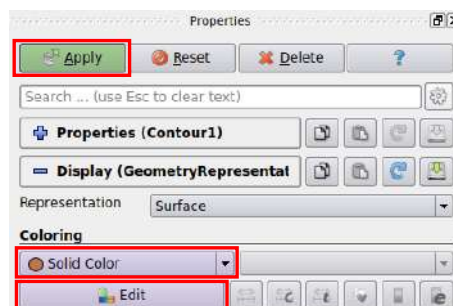


Figure 18.9: Changing contour line colors in the **Coloring** section.

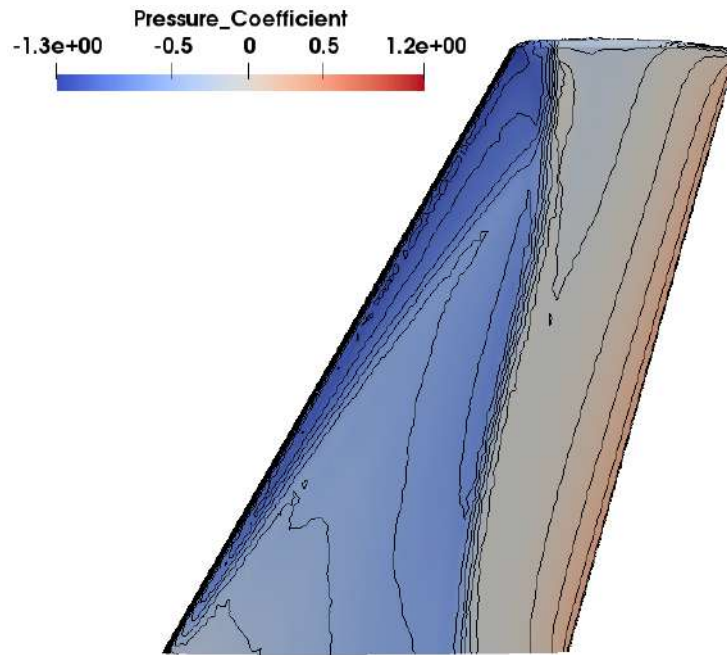
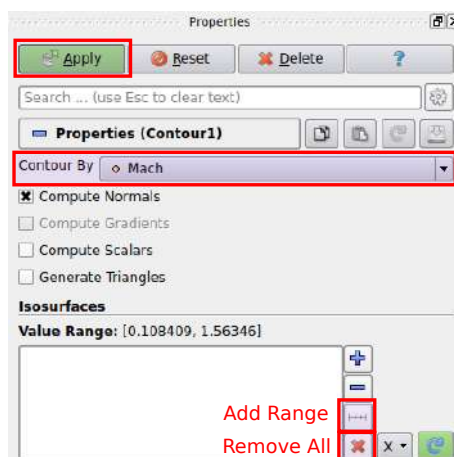
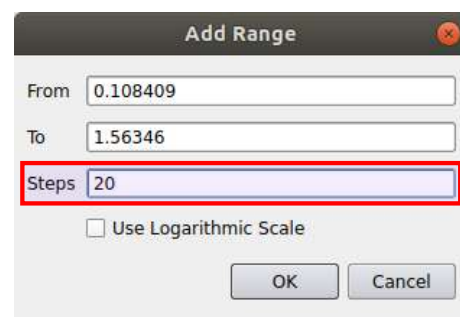


Figure 18.10: Pressure coefficient contours superimposed with contour lines for the ONERA M6 wing.

- 14) To display Mach number contours, similar to Figure 18.4, click on `surface_flow.vtk` in the **Pipeline Browser**.
- 15) Under **Coloring** in the **Display** section, select *Mach* from the drop-down menu.
- 16) Click on "Contour1" in the **Pipeline Browser**.
- 17) As shown in Figure 18.11a, under the **Properties** tab, select *Mach* from the **Contour By** drop-down menu.
- 18) Click on the **Remove All** icon to remove the previous range used for the pressure coefficient.
- 19) Similar to Figure 18.11b, click on the **Add Range** icon and set **Steps** to 20.
- 20) Next, click on **OK** and **Apply** to show the plot. Now the Mach number contours on the wing surface should be shown, similar to Figure 18.12.



(a) Define a new range



(b) Add the new range

Figure 18.11: How to define a new range for the contour lines.

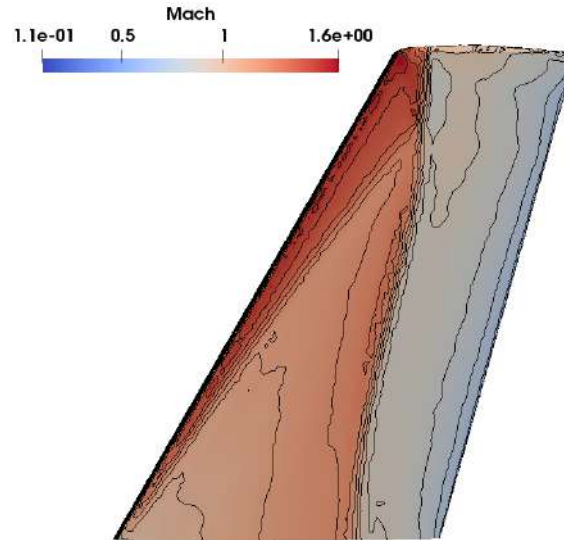


Figure 18.12: Mach number contour with superimposed by lines.

18.0.4 Comparison of Convergence Rates

We will now explore the rate of convergence of SU2 for this problem. To do this, we are going to consider the residual of the solution at each iteration as well as the aerodynamic loads in order to see how they change with respect to the number of iterations. Therefore, you can take the following steps:

- 14) Launch any suitable plotting or spreadsheet software.
- 15) Open the file `history.vtk` that was generated by SU2.
- 16) Choose the file type that best describes your data, and tell your software to use *Tab*, *Semicolon*, and *Comma* as possible delimiters (Figure 18.13).

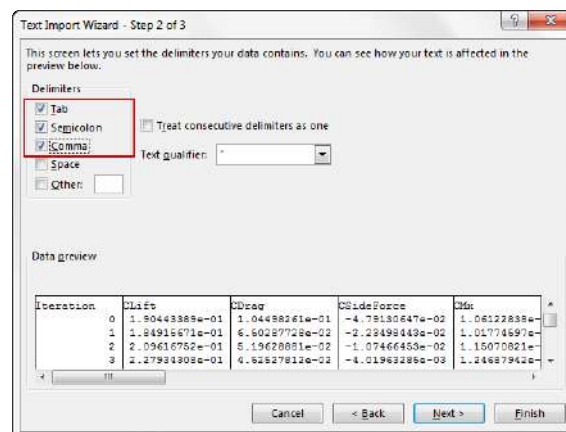


Figure 18.13: Import the `.vtk` file into plotting or spreadsheet software.

As shown in Figure 18.14, the first column (Column A) shows the iteration number. The lift coefficient (C_L) and the drag coefficient (C_D) are displayed in the second and third columns, respectively (Column B and Column C). Finally, the residuals can also be examined to check the rate of convergence (Column L to Column P in Figure 18.15).

	A	B	C	D	E	F	G	H	I
Iteration	CLift	CDrag	CSideForc	CMx	CMy	CMz	CFx	CFy	CFz
0	1.90E-01	1.04E-01	-4.79E-02	1.06E-01	-6.72E-02	-5.59E-02	9.42E-02	-4.79E-02	
1	1.85E-01	6.50E-02	-2.23E-02	1.02E-01	-5.99E-02	-3.09E-02	5.51E-02	-2.23E-02	
2	2.10E-01	5.20E-02	-1.07E-02	1.15E-01	-6.46E-02	-2.02E-02	4.07E-02	-1.07E-02	
3	2.28E-01	4.53E-02	-4.02E-03	1.25E-01	-6.67E-02	-1.34E-02	3.30E-02	-4.02E-03	
4	2.39E-01	3.84E-02	1.44E-03	1.30E-01	-6.70E-02	-7.01E-03	2.56E-02	1.44E-03	
5	2.48E-01	3.28E-02	5.38E-03	1.35E-01	-6.69E-02	-1.89E-03	1.95E-02	5.38E-03	
6	2.54E-01	2.86E-02	7.99E-03	1.38E-01	-6.67E-02	1.87E-03	1.50E-02	7.99E-03	
7	2.59E-01	2.54E-02	9.77E-03	1.40E-01	-6.64E-02	4.70E-03	1.15E-02	9.77E-03	
8	2.64E-01	2.28E-02	1.11E-02	1.42E-01	-6.63E-02	6.91E-03	8.72E-03	1.11E-02	
9	2.68E-01	2.07E-02	1.20E-02	1.43E-01	-6.65E-02	8.68E-03	6.37E-03	1.20E-02	
10	2.72E-01	1.89E-02	1.28E-02	1.45E-01	-6.68E-02	1.01E-02	4.34E-03	1.28E-02	
11	2.75E-01	1.73E-02	1.34E-02	1.46E-01	-6.73E-02	1.13E-02	2.57E-03	1.34E-02	
12	2.79E-01	1.59E-02	1.39E-02	1.48E-01	-6.77E-02	1.23E-02	1.00E-03	1.39E-02	
13	2.82E-01	1.47E-02	1.43E-02	1.49E-01	-6.81E-02	1.32E-02	-3.74E-04	1.43E-02	
14	2.84E-01	1.36E-02	1.46E-02	1.50E-01	-6.84E-02	1.38E-02	-1.57E-03	1.46E-02	
15	2.86E-01	1.27E-02	1.48E-02	1.51E-01	-6.86E-02	1.44E-02	-2.59E-03	1.48E-02	

Figure 18.14: Columns in history.vtk

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Iteration	CLift	CDrag	CSideForc	CMx	CMy	CMz	CFx	CFy	CFz	CL/CD	Res_Flow	Res_Flow	Res_Flow	Res_Flow	Res_Flow
0	1.90E-01	1.04E-01	-4.79E-02	1.06E-01	-6.72E-02	-5.59E-02	9.42E-02	-4.79E-02	1.96E-01	1.82E+00	-2.66E+00	-2.06E-01	-1.11E+01	-1.48E+00	2.86E+00
1	1.85E-01	6.50E-02	-2.23E-02	1.02E-01	-5.99E-02	-3.09E-02	5.51E-02	-2.23E-02	1.88E-01	2.84E+00	-2.30E+00	3.44E-01	7.66E-03	2.23E-01	3.26E+00
2	2.10E-01	5.20E-02	-1.07E-02	1.15E-01	-6.46E-02	-2.02E-02	4.07E-02	-1.07E-02	2.12E-01	4.03E+00	-2.24E+00	3.90E-01	2.43E-01	4.16E-01	3.29E+00
3	2.28E-01	4.53E-02	-4.02E-03	1.25E-01	-6.67E-02	-1.34E-02	3.30E-02	-4.02E-03	2.30E-01	5.04E+00	-2.17E+00	4.34E-01	3.77E-01	5.43E-01	3.34E+00
4	2.39E-01	3.84E-02	1.44E-03	1.30E-01	-6.70E-02	-7.01E-03	2.56E-02	1.44E-03	2.41E-01	6.23E+00	-2.14E+00	4.60E-01	4.50E-01	6.37E-01	3.37E+00
5	2.48E-01	3.28E-02	5.38E-03	1.35E-01	-6.69E-02	-1.89E-03	1.95E-02	5.38E-03	2.49E-01	7.56E+00	-2.13E+00	4.46E-01	4.85E-01	7.01E-01	3.37E+00
6	2.54E-01	2.86E-02	7.99E-03	1.38E-01	-6.67E-02	1.87E-03	1.50E-02	7.99E-03	2.55E-01	8.88E+00	-2.12E+00	4.17E-01	4.95E-01	7.35E-01	3.36E+00
7	2.59E-01	2.54E-02	9.77E-03	1.40E-01	-6.64E-02	4.70E-03	1.15E-02	9.77E-03	2.60E-01	1.02E+01	-2.12E+00	3.93E-01	4.92E-01	7.45E-01	3.36E+00
8	2.64E-01	2.28E-02	1.11E-02	1.42E-01	-6.63E-02	6.91E-03	8.72E-03	1.11E-02	2.65E-01	1.16E+01	-2.12E+00	3.66E-01	4.82E-01	7.34E-01	3.35E+00
9	2.68E-01	2.07E-02	1.20E-02	1.43E-01	-6.65E-02	8.68E-03	6.37E-03	1.20E-02	2.69E-01	1.29E+01	-2.14E+00	3.25E-01	4.66E-01	7.08E-01	3.33E+00
10	2.72E-01	1.89E-02	1.28E-02	1.45E-01	-6.68E-02	1.01E-02	4.34E-03	1.28E-02	2.72E-01	1.44E+01	-2.16E+00	2.74E-01	4.45E-01	6.72E-01	3.30E+00
11	2.75E-01	1.73E-02	1.34E-02	1.46E-01	-6.73E-02	1.13E-02	2.57E-03	1.34E-02	2.76E-01	1.59E+01	-2.17E+00	2.26E-01	4.17E-01	6.31E-01	3.29E+00
12	2.79E-01	1.59E-02	1.39E-02	1.48E-01	-6.77E-02	1.23E-02	1.00E-03	1.39E-02	2.79E-01	1.75E+01	-2.19E+00	1.95E-01	3.82E-01	5.90E-01	3.27E+00
13	2.82E-01	1.47E-02	1.43E-02	1.49E-01	-6.81E-02	1.32E-02	-3.74E-04	1.43E-02	2.82E-01	1.92E+01	-2.20E+00	1.79E-01	3.41E-01	5.51E-01	3.25E+00
14	2.84E-01	1.36E-02	1.46E-02	1.50E-01	-6.84E-02	1.38E-02	-1.57E-03	1.46E-02	2.84E-01	2.09E+01	-2.23E+00	1.70E-01	2.95E-01	5.14E-01	3.22E+00

Figure 18.15: Residuals in history.vtk

Now, we can plot the predicted lift coefficient, drag coefficient, and density residual against the number of iterations to visualize when the solver has converged to a steady solution. Figure 18.16 and Figure 18.17 show plots for the aerodynamic loads and residual as a function of the number of iterations, respectively, for the default case of no multigrid, MGLEVEL=0.

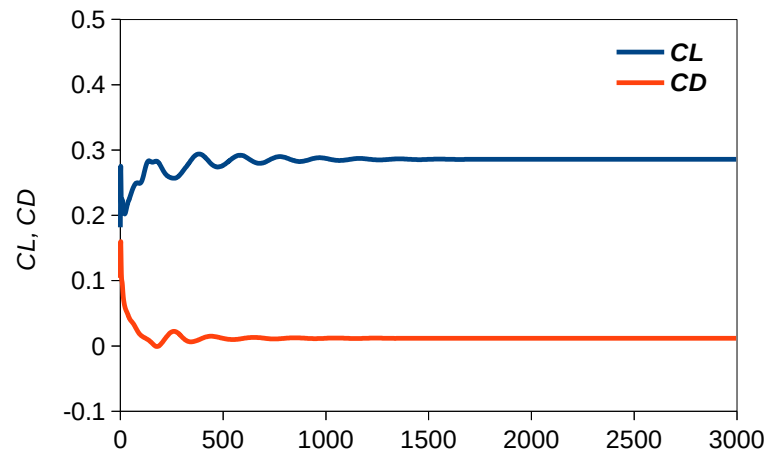


Figure 18.16: Lift and Drag coefficients versus the number of iterations.

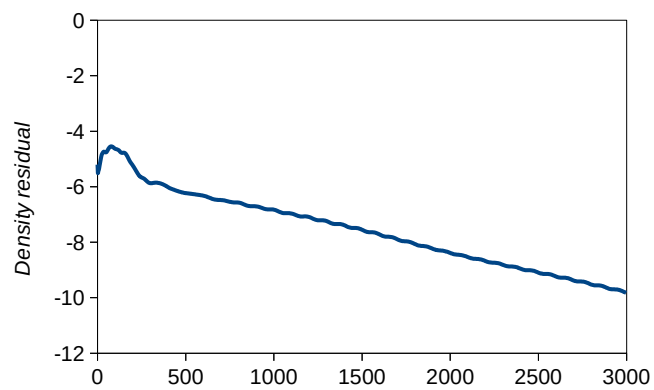


Figure 18.17: Density residual versus the number of iterations.

Questions

1. Run the simulation without multigrid ($\text{MGLEVEL}=0$).
 - (a) Follow the procedure in the guideline document to run the simulation and record the number of iterations and wall-clock time for the simulation to converge.
 - (b) Record the lift and drag coefficients.
 - (c) Plot the surface pressure coefficient with coloured contours and contour lines.
 - (d) Plot the surface Mach number with coloured contours and contour lines.
 - (e) Plot the value of C_L , C_D and the density residual vs. the number of iterations.
2. Modify the configuration file in the multigrid folder ($\text{MGLEVEL}=3$, $\text{MGCYCLE}=\text{W_CYCLE}$) and repeat Q.1 using multigrid.
3. Compare the results in part (a)-(e) of Q.1 and Q.2. In what way is the use of multigrid advantageous for this test case?

19. Laminar Cylinder

Required Files



Use the following links to download the same version of SU2 for Windows ([click here](#)) or Mac ([click here](#)), and the required configuration and mesh files for the steady case ([click here](#)), unsteady case ([click here](#)), and the reference datasets ([click here](#)).



Use the following links to download the same version of Paraview for Windows ([click here](#)) or Mac ([click here](#)).

Problem Description

In this tutorial, we will explain how to simulate external viscous flow past a 2D cylinder. To simulate this, the Navier-Stokes equations will be solved in both steady and unsteady forms, depending on the chosen Reynolds number. For this type of flow, the wake is expected to remain steady up to a Reynolds number of around 46. At higher Reynolds numbers, the flow becomes unsteady with oscillating vortices being shed in its wake. This is known as a Von-Karman vortex street, which is a well-known phenomenon in fluid mechanics. The computational domain for this tutorial uses an O-topology mesh with the cylinder in the center of the domain. The outer boundary is situated at a radial distance of $15D$, where D is cylinder diameter. The resulting mesh is composed of 26,192 triangular elements. The elements around the surface of the cylinder are refined to resolve the viscous boundary layer in this region. For this example, the flow specifications are provided as follows:

- Pressure = 101,325Pa
- Temperature = 273.15K
- Mach number = 0.1
- Angle of attack = 0 degrees
- Reynolds number = 40
- Characteristic length = 1m

This tutorial has two parts: Flow Solution and Post-processing. In the first part, we will explain how to manage the prerequisite files and settings, and how to run the CFD simulation using SU2. In the second part, we explain how to use Paraview to visualize the data obtained from SU2.

Flow solution

To run this simulation, SU2 needs two files: a configuration file (`.cfg`) and a mesh file (`.su2`). Links to the required files and executables are provided at the start of this tutorial. The files include:

1. `lam_cylinder.cfg` as a configuration file.
2. `mesh_cylinder_lam.su2` as a mesh file.

The next step is to copy these two files into the same directory as the SU2 executable. To run the simulation, open a terminal window and enter the following commands:

Windows	\$ cd "where you saved the package" \$ SU2_CFD.exe lam_cylinder.cfg
Mac	\$ cd "where you saved the package" \$./SU2_CFD lam_cylinder.cfg

SU2 will begin solving the problem, and will print out the residuals at every iteration until the specified convergence criteria is reached. The computational time for this case is highly dependant on the computer's performance. However, the run time is expected to be about 15 minutes on average. After converging, the following output files will be generated and saved in the SU2 folder:

- `flow.vtk`: The flow solution on the entire domain.
- `force_breakdown.dat`: Forces and moment on the cylinder.
- `history.vtk`: Convergence history.
- `restart_flow.dat`: Restart file.
- `surface_flow.vtk`: The flow solution on the cylinder surface.
- `surface_flow.csv`: A comma separated value file of the flow solution on the cylinder.

Please keep in mind that every time you run SU2, the output data will be overwritten. Hence, before launching a new simulation you should backup your files in another directory.

Post-processing

In this section, we will explain how to use Paraview to visualize the data produced by SU2.

19.0.1 Load the Solution File:

In order to import data to Paraview, you can take the following steps:

- 1) Launch Paraview.
- 2) Go to **File** → **Open**, and then select the `flow.vtk` file.
- 3) On the left-hand side of the Paraview window you will see the file appear under **builtin** in the **Pipeline Browser**.
- 4) Now press the **Apply** button in the **Properties** tab, directly under the **Pipeline Browser**. The solution file is now loaded by Paraview and is ready to be visualized (Figure 19.1).

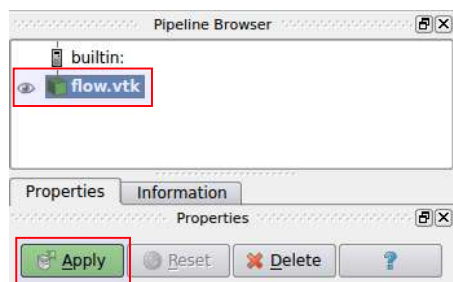


Figure 19.1: Loading `flow.vtk` file in the **Pipeline Browser**.

19.0.2 Visualize the Mesh

In order to display the mesh, as shown in Figure 19.2, select *Solid Color* with *Wireframe* in the toolbar. Then, you can zoom-in to see mesh around the surface of the cylinder, as shown in Figure 19.3. The mesh around the cylinder is unstructured, and the elements are clustered around the cylinder to be able to capture the boundary layer region properly.

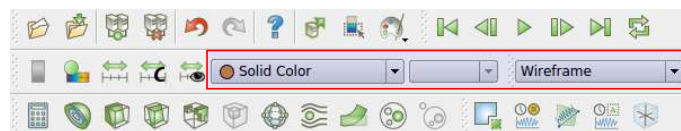


Figure 19.2: How to display the cylinder mesh.

19.0.3 Visualize Pressure Contour and Much Number Contour

To visualize pressure contours in the domain, you can take the following steps:

- 1) Activate `flow.vtk` by clicking on it in the **Pipeline Browser**.
- 2) Go to **Display** in **Properties** tab.
- 3) Under the **Coloring** section, select *Pressure* from the drop-down menu (Figure 19.4). Contours of the pressure coefficient will now be displayed, similar to Figure 19.5.
- 4) To change the color range used for pressure, you can also click on **Edit** under the same **Coloring** section.
- 5) To add contour lines to the plot, click again on the `flow.vtk` file in the **Pipeline Browser**.
- 6) Click on the **Contour** icon (Figure 19.6).

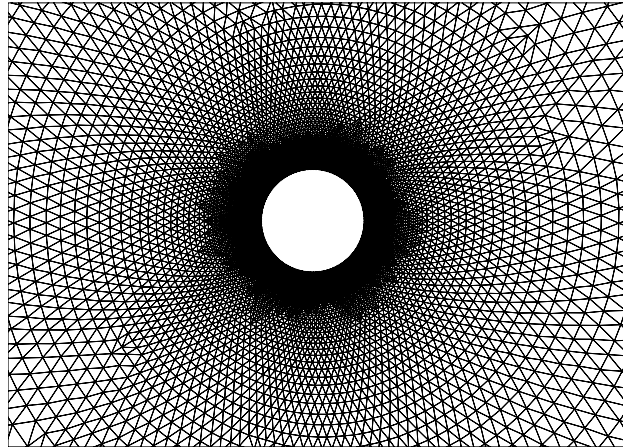


Figure 19.3: The cylinder mesh.



Figure 19.4: How to display contours by selecting the variable.



Figure 19.6: The **Contour** icon in the toolbar.

- 7) Similar to Figure 19.7, you should now see *Contour1* under the `flow.vtk` file in the **Pipeline Browser**.
- 8) As shown in Figure 19.8a, in the **Properties** section select *Pressure* from the **Contour By** drop-down menu.
- 9) You can use the **Remove All** icon to remove the default range.
- 10) Then, click on the **Add Range** icon to make a new range based on your preferences.
- 11) Similar to Figure 19.8b, set **Steps** to 20, and then, click on **OK** and **Apply**, respectively.
- 12) To change the color of the contour lines, activate *Contour1* by clicking on it in **Pipeline Browser**.
- 13) Similar to Figure 19.4, you can go to **Edit** under the **Coloring** section and select white. Finally, the pressure contours superimposed with appropriate contour lines should be visible, similar to Figure 19.9. Keep in mind that to display both contour and contour lines, the eye beside each item in the **Pipeline Browser** should be activated (Figure 19.10)
- 14) To display the Mach number contours, click on `flow.vtk` again in the **Pipeline Browser**.
- 15) As shown in Figure 19.11, under the **Coloring** section in **Display**, select *Mach* from the drop-down menu.
- 16) Click on *Contour1* in the **Pipeline Browser**.
- 17) Similar to Figure 19.12a, in **Properties** select *Mach* from the **Contour By** drop-down menu.

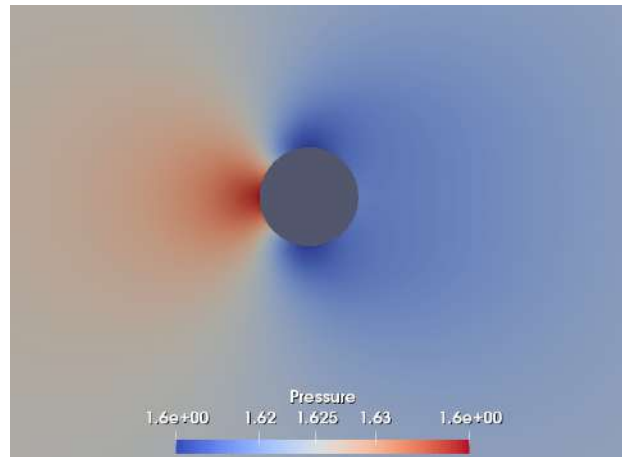
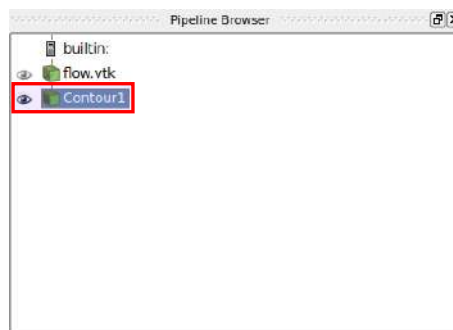
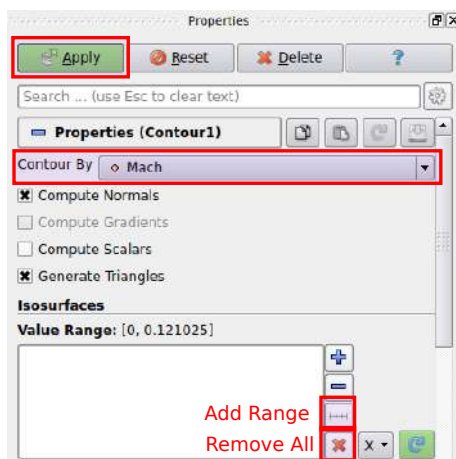


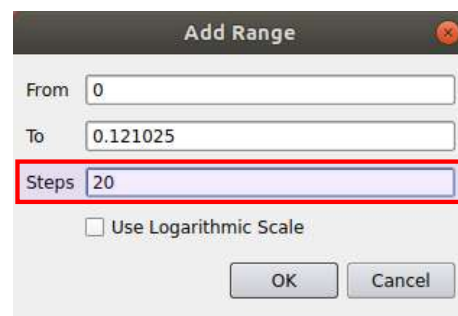
Figure 19.5: Pressure contours around the cylinder.

Figure 19.7: Adding *Contour1* to the **Pipeline Browser**.

- 18) Click on **Remove All** to remove the previous value range used for pressure.
- 19) Click on **Add Range** and set the value of **Steps** to 20.
- 20) Click on **OK** and **Apply**, respectively (Figure 19.12b). Mach number contours around the cylinder should now be visible, similar to Figure 19.13.

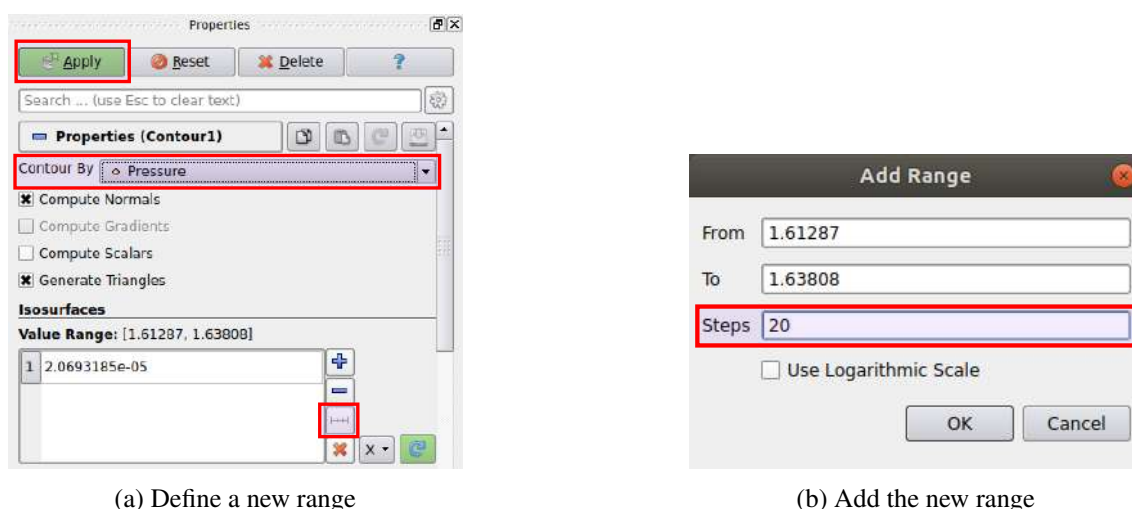


(a) Define a new range



(b) Add the new range

Figure 19.12: How to define a new range for the contour lines.



(a) Define a new range

(b) Add the new range

Figure 19.8: How to define a new range for the contour lines.

19.0.4 Streamlines and Separation Length

Streamlines show the path that a fluid element will follow, when released. Streamlines can help with visualizing flow separation and the wake region behind bluff bodies, such as the cylinder used in this tutorial. In this case, streamlines can be used to visualize the separation bubble behind the cylinder. To do this, you can take the following steps:

- 1) Activate `flow.vtk` by clicking on it in the **Pipeline Browser**.
- 2) As shown in Figure 19.14, click on the **Calculator** icon, which allows you to define new variables and add them to the list of variables.
- 3) Click on *Calculator1* in the **Pipeline Browser** and, as shown in Figure 19.15, rename the **Result Array Name** to *Velocity*.
- 4) Following Figure 19.15, write the contribution of each velocity component in the equation box. Keep in mind that dividing momentum by density results the corresponding fluid velocity component.
- 5) Then, click on **Apply** to proceed to the next step.
- 6) As shown in Figure 19.16, *Calculator1* now appears in the **Pipeline Browser** as a subset of `flow.vtk`.
- 7) Now click again on *Calculator1* in the **Pipeline Browser**.
- 8) Click on the **Stream Tracer** icon in the toolbar (Figure 19.17).
- 9) Click on *StreamTracer1* in the **Pipeline Browser**.
- 10) As shown in Figure 19.18, in the **Properties** menu select *Velocity* from **Vectors** drop-down menu.
- 11) Select *High Resolution Line Source* from the **Seed Type** drop-down menu.
- 12) Under **Line Parameters**, set **Point1** and **Point2** as (-15,0,0) and (15,0,0), respectively. These two points define the endpoints of a line along which streamlines will be generated.
- 13) As shown in Figure 19.19, under **Coloring** in the **Display** section select *Solid Color* from **Edit**, and then choose white color for the streamlines
- 14) Click on the check-box beside **Data Axis Grid** to display the axis range in the plot.
- 15) Hide all items except `flow.vtk` and *StreamTracer1* by deactivating the eye icon beside each of them (Figure 19.20).

Finally, Figure 19.21 shows the Mach number contours with superimposed streamlines. As mentioned previously, the diameter of the cylinder is taken to be 1m. From Figure 19.21, you can approximate the separation length to be around 2m behind the cylinder at the chosen $Re = 40$.

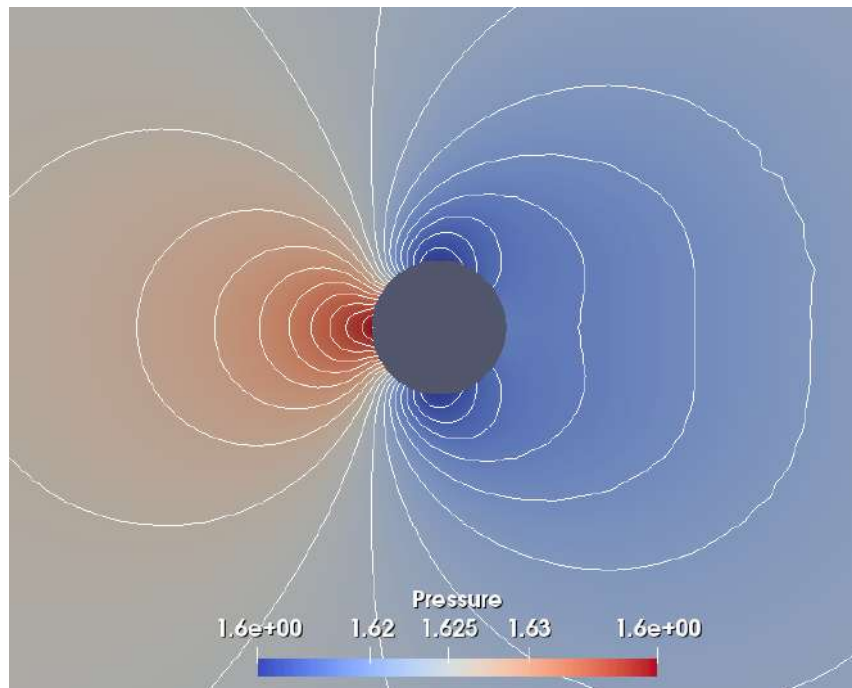


Figure 19.9: Pressure contours superimposed by contour lines around the cylinder.

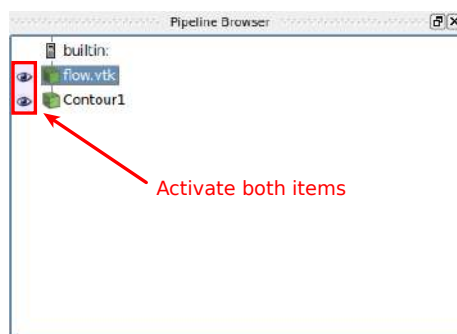


Figure 19.10: How to display contour and contour lines at the same time in the plot.

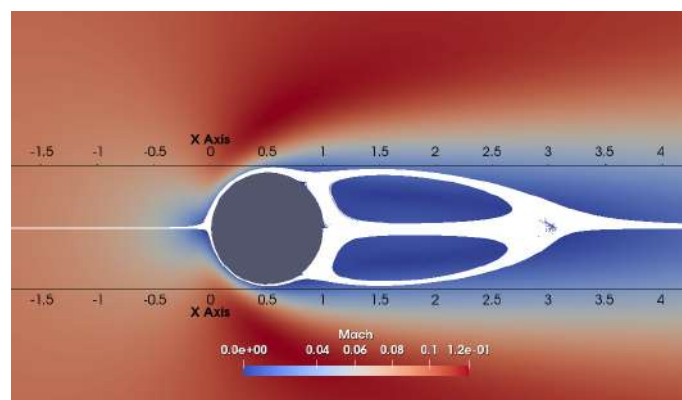


Figure 19.21: Mach number contours superimposed with streamlines.

The length of the separated bubble can also be calculated more precisely. To do this we will plot the streamwise velocity component along the computational domain center-line, and measure

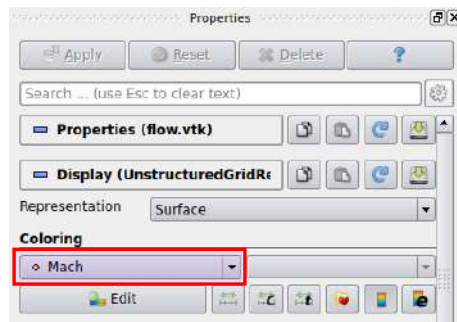


Figure 19.11: How to change the color for contour lines of Mach number.

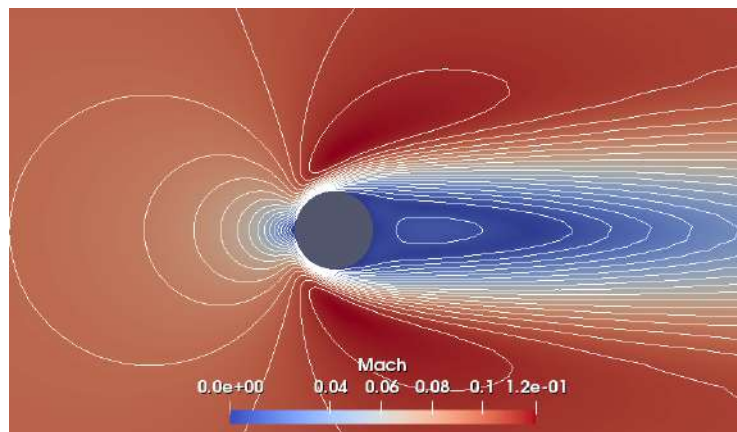
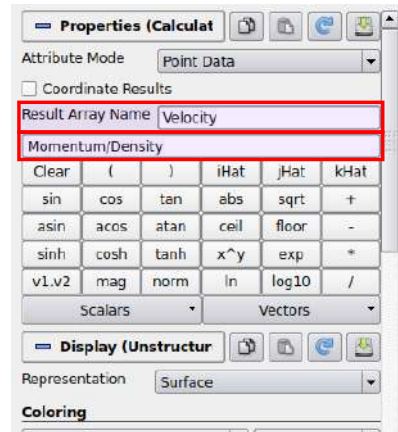
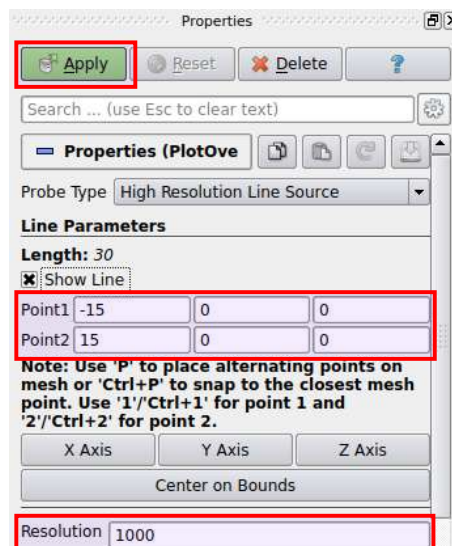


Figure 19.13: Mach number contour with superimposed by lines.

the length of the separation bubble, which is defined as the region of reversed flow in the wake. Therefore, the separation bubble length can be found as the distance between the surface of the cylinder and the location where the *Velocity_X* component switches from negative to positive. To get started, you can take the following steps:

- 1) In the **Pipeline Browser** click on *Calculator1*.
- 2) Go to the **Filters** → **Data Analysis** → **Plot Over Line** option (Figure 19.22).
- 3) Activate *PlotOverLine1* by clicking on it in the **Pipeline Browser**.
- 4) As shown in Figure 19.23, under **Line Parameters** in the **Properties** menu set **Point1** and **Point2** as (-15,0,0) and (15,0,0), respectively. Please note that this line is different from the previous line we defined to generate streamlines.
- 5) Set the **Resolution** to 10,000, and then click on **Apply**.

Figure 19.14: How to define new variables using the **Calculator**.Figure 19.15: How to use a **Calculator** to compute the velocity.Figure 19.23: Settings for **Plot Over Line**.

6) As shown in Figure 19.24, in the **Display** panel select *Point_X* from the **X Array Name** drop-down menu.

7) Under the **Series Parameters**, unselect all variables except for *Velocity_X*.

A figure similar to Figure 19.25 will now display the velocity profile over the chosen line along the (*X Axis*). As shown in Figure 19.25, the separation length, *L*, can be measured when *Velocity_X* approaches zero in the wake of the cylinder.

19.0.5 Shedding Frequency

When the Reynolds number is high enough, the wake behind the cylinder becomes unsteady, and vortex shedding occurs. The frequency of the wake oscillations is called the “shedding frequency.” To explore this unsteady case, open the `history.vtk` in an appropriate plotting program. As

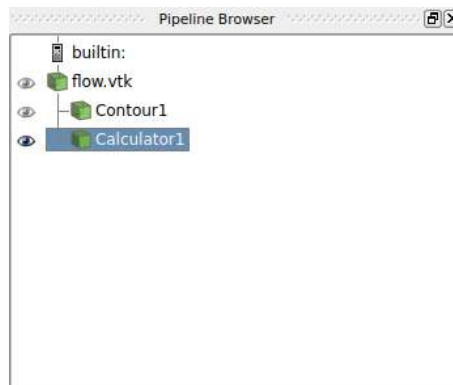


Figure 19.16: *Calculator1* as a subset of `flow.vtk` in the **Pipeline Browser**.



Figure 19.17: How to add streamlines using **Stream Tracer**.

shown in Figure 19.26, the first column in the file shows the number of iterations. Note that the physical time can be computed by multiplying the iteration number with the time-step size, which is $\Delta t = 0.001\text{s}$ for the unsteady case as set in the configuration file. Plotting C_L or C_D as a function of time demonstrates that the shedding amplitude grows, and then remains quasi-steady. The non-dimensional Strouhal number of this vortex shedding can now be calculated from $St = \frac{fD}{U}$, where f , D and U are the shedding frequency, cylinder diameter, and free-stream velocity, respectively. The shedding frequency can be found manually by determining the time interval between vortex shedding events in the C_L time history, within the quasi-steady region.

Questions

1. Run the laminar flow over a cylinder as per the tutorial case but at $Re = 10, 20$, and 40 .
 - (a) Plot Mach contours with streamlines for $Re = 10, 20$, and 40
 - (b) Plot the x -velocity component vs the x -coordinate beginning at $x = 1$ (location of rear surface of cylinder)
 - (c) Calculate L/D (non-dimensional separation length) for the three Re cases and compare with the experimental L/D data provided at the start of this tutorial [3]. Also include the C_D values for the three Re cases from the simulation results. Comment on the effect of Re on the non-dimensional separation length L/D and C_D .
2. Download the unsteady configuration files provided at the start of this tutorial for $Re = 150$.
 - (a) Plot the C_L and C_D versus time.
 - (b) Calculate the amplitudes ΔC_L , ΔC_D , and the Strouhal number, and compare your results with experimental data provided at the start of this tutorial [8].

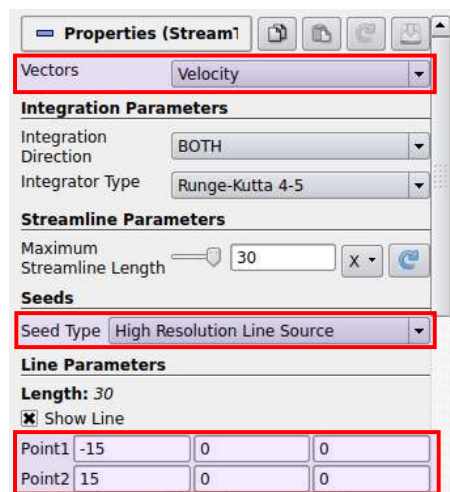


Figure 19.18: Streamline settings in the **Stream Tracer** panel.

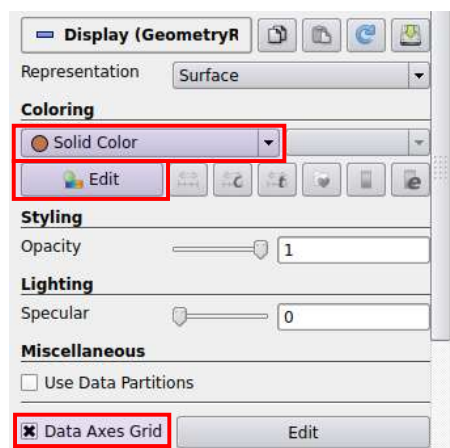


Figure 19.19: Changing the streamline colors in the **Display** panel.

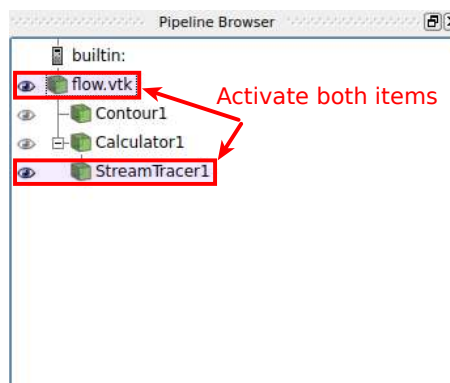


Figure 19.20: Activating both `flow.vtk` and `StreamTracer1` in the **Pipeline Browser**.

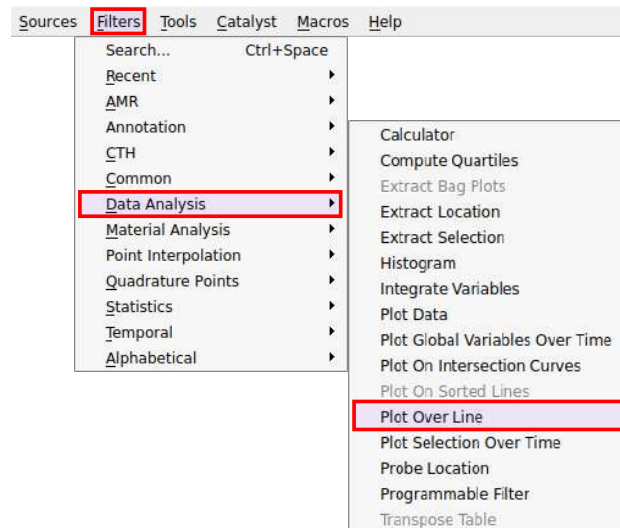


Figure 19.22: How to plot over an arbitrary line.

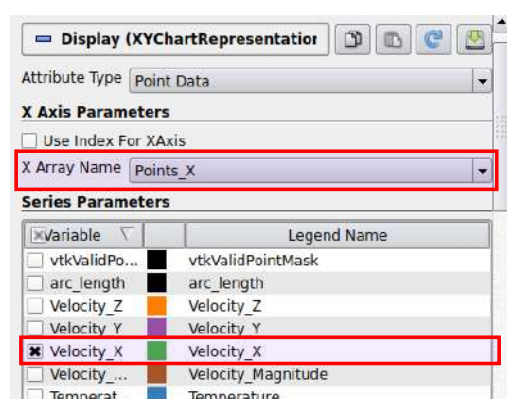


Figure 19.24: How to Plot *Velocity_X* versus *X Axis*.

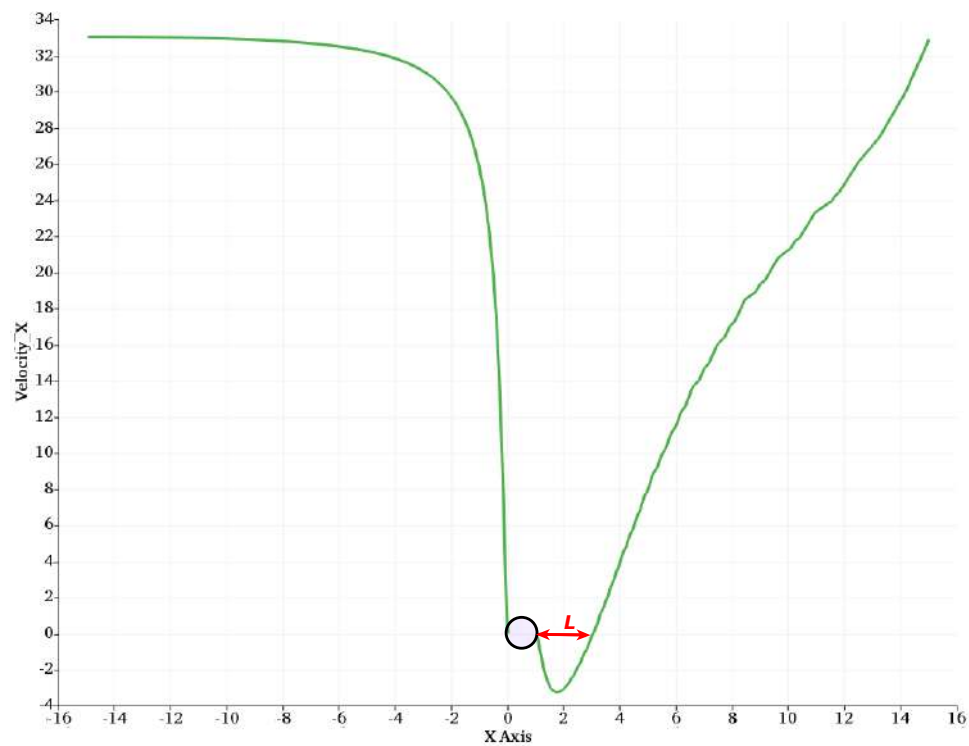


Figure 19.25: *Velocity_X* versus *X Axis*, and the length of the separation bubble.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Iteration	CLift	CDrag	CSideForc	CMx	CMy	CMz	CFx	CFy	CFz	CL/CD	HeatFlux_	HeatFlux_	Res_Flow
2	0	9.31E-04	2.70E+01	0.00E+00	0.00E+00	0.00E+00	-8.01E-04	2.70E+01	9.31E-04	0.00E+00	3.45E-05	-6.95E-01	8.56E-01	-1.03E+0
3	1	5.43E-04	1.91E+01	0.00E+00	0.00E+00	0.00E+00	-3.23E-04	1.91E+01	5.43E-04	0.00E+00	2.84E-05	-3.76E-01	5.66E-01	-1.08E+0
4	2	5.59E-04	1.21E+01	0.00E+00	0.00E+00	0.00E+00	-3.46E-04	1.21E+01	5.59E-04	0.00E+00	4.61E-05	-2.89E-01	2.93E-01	-1.09E+0
5	3	5.66E-05	6.92E+00	0.00E+00	0.00E+00	0.00E+00	-4.63E-04	6.92E+00	5.66E-05	0.00E+00	8.18E-06	-2.49E-01	2.65E-01	-1.11E+0
6	4	-2.45E-04	3.38E+00	0.00E+00	0.00E+00	0.00E+00	-4.74E-04	3.38E+00	-2.45E-04	0.00E+00	-7.26E-05	-2.11E-01	2.35E-01	-1.13E+0
7	5	-4.05E-04	1.28E+00	0.00E+00	0.00E+00	0.00E+00	-4.28E-04	1.28E+00	-4.05E-04	0.00E+00	-3.17E-04	-1.79E-01	2.12E-01	-1.16E+0
8	6	-4.62E-04	2.36E-01	0.00E+00	0.00E+00	0.00E+00	-3.60E-04	2.36E-01	-4.62E-04	0.00E+00	-1.95E-03	-1.53E-01	1.93E-01	-1.18E+0
9	7	-4.37E-04	-1.07E-01	0.00E+00	0.00E+00	0.00E+00	-2.89E-04	-1.07E-01	-4.37E-04	0.00E+00	4.09E-03	-1.34E-01	1.76E-01	-1.18E+0
10	8	-3.66E-04	-4.75E-02	0.00E+00	0.00E+00	0.00E+00	-2.26E-04	-4.75E-02	-3.66E-04	0.00E+00	7.72E-03	-1.20E-01	1.63E-01	-1.18E+0
11	9	-2.80E-04	2.00E-01	0.00E+00	0.00E+00	0.00E+00	-1.74E-04	2.00E-01	-2.80E-04	0.00E+00	-1.40E-03	-1.10E-01	1.54E-01	-1.19E+0

Figure 19.26: Time history of flow variables.

20. Turbulent ONERA M6

Required Files



Use the following links to download the same version of SU2 for Windows ([click here](#)) or Mac ([click here](#)), the required configuration and mesh files ([click here](#)), and reference dataset ([click here](#)).



Use the following links to download the same version of Paraview for Windows ([click here](#)) or Mac ([click here](#)).

Problem Description

In this tutorial, we will demonstrate how to simulate transonic viscous flow past a three-dimensional wing. We will use the ONERA M6 again, and detailed information is given in the previous tutorial for inviscid flow in Chapter 18. We will use the Reynolds-Averaged Navier-Stokes (RANS) equations and compare the Spalart-Almaras (SA) and $k - \omega$ Shear Stress Transport (SST) turbulence models for turbulent eddy viscosity. The computational domain uses a structured mesh with 43,008 hexahedral elements. Note that this mesh is relatively coarse to allow for relatively short computation time. The flow specifications are provided as follows:

- Temperature = 288.15 K
- Mach number = 0.8395
- Angle of attack = 3.06 degrees
- Reynolds length = 0.64607 m

This tutorial has two parts: Flow Solution and Post-processing. In the first part, we will explain how to manage the prerequisite files and settings and how to run the CFD simulation using SU2. In the second part, we explain how to use Paraview to visualize the data obtained from SU2.

Flow solution

To run this simulation, SU2 needs two files: a configuration file (`.cfg`) and a mesh file (`.su2`). Links to the required files and executables are provided at the start of this tutorial. The files include

1. `turb_ONERAM6.cfg` as a configuration file.
2. `mesh_ONERAM6_turb_hexa_43008.su2` as a mesh file.

The next step is to copy these two files into the same directory as the SU2 executable. To run the simulation, open a terminal window and enter the following commands:

Windows	\$ cd "where you saved the package" \$ SU2_CFD.exe turb_ONERAM6.cfg
Mac	\$ cd "where you saved the package" \$./SU2_CFD turb_ONERAM6.cfg

SU2 will begin solving the problem and will print out the residuals at every iteration until the specified convergence criterion is reached. The computational time for this case is highly dependant on the computer's performance. However, the run time is expected to be about 4 hours on average. After this, the following output files should be generated and saved in the SU2 folder:

- `flow.vtk`: The flow solution on the entire domain.
- `force_breakdown.dat`: Forces and moment on the wing.
- `history.vtk`: Convergence history.
- `restart_flow.dat`: Restart file.
- `surface_flow.vtk`: The flow solution on the wing surface.
- `surface_flow.csv`: A comma separated value file of the flow solution on the wing.

Please keep in mind that every time you run SU2, the output data will be overwritten. Hence, before launching a new simulation you should backup your files in another directory.

Post-processing

In this section we demonstrate how to use Paraview to visualize the data produced by SU2.

20.0.1 Load the Solution File:

In order to import data to Paraview, you can take the following steps:

- 1) Launch Paraview.
- 2) Go to **File** → **Open**, and then select the `flow.vtk` file.
- 3) On the left-hand side of the Paraview window you will see the file appear under **builtin** in the **Pipeline Browser**.
- 4) Now press the **Apply** button in the **Properties** tab, directly under the **Pipeline Browser**. The solution file is now loaded by Paraview and is ready to be visualized (Figure 20.1).



Figure 20.1: Loading `surface_flow.vtk` file in the **Pipeline Browser**.

20.0.2 Visualize the Mesh

In order to view the mesh, as shown in Figure 20.2, select *Solid Color* with *Wireframe* in the toolbar. Then, you can zoom-in to see mesh around the airfoil, as shown in Figure 20.3. You can see that this mesh uses structured elements, and is highly refined in the boundary layer, leading edge, and trailing edge regions.

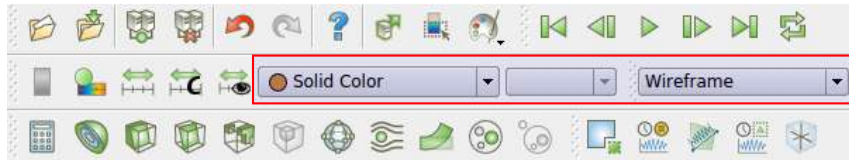


Figure 20.2: How to display mesh in computational domain.

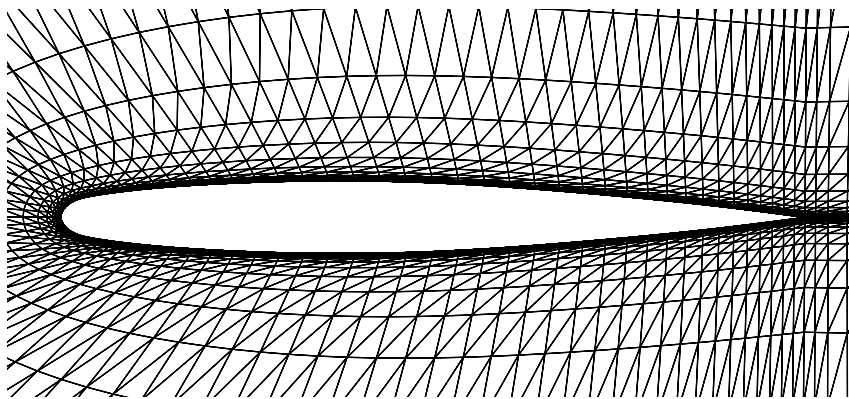


Figure 20.3: Structured mesh for ONERA M6 wing. The mesh around the surface is refined to capture the boundary layer.

20.0.3 Visualize Pressure Coefficient at Different Stations

In this section, we will explain how to plot the pressure coefficient on the surface of the wing as a function of chord-wise position at different span-wise stations along the wing. To get started, you can take the following steps:

- 1) Activate `surface_flow.vtk` by clicking on it in the **Pipeline Browser**.
- 2) Click on the **Slice** icon in the toolbar, as shown in Figure 20.4. This will allow you to slice a plane through the wing, to plot the pressure coefficient (C_p) at a specific station or location along the span.
- 3) As shown in Figure 20.5, click on *Y Normal*, and under **Plane Parameters** specify the y-coordinate location in the **Origin** field where we want the slice to be taken.
- 4) For this example we select (0.5705415, 0.6081815, 0) as a suitable value for **Origin**, which is approximately in the middle of the wing (Figure 20.6).
- 5) Click on **Apply** after specifying the y-coordinate.



Figure 20.4: How to slice a 3D computational domain.

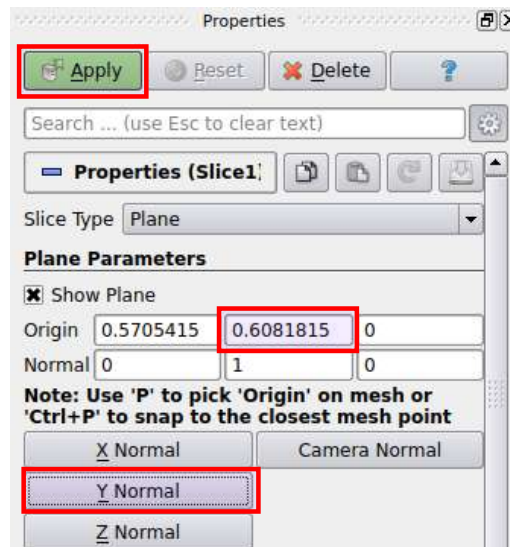


Figure 20.5: Setting for **Slice** in **Properties** panel.

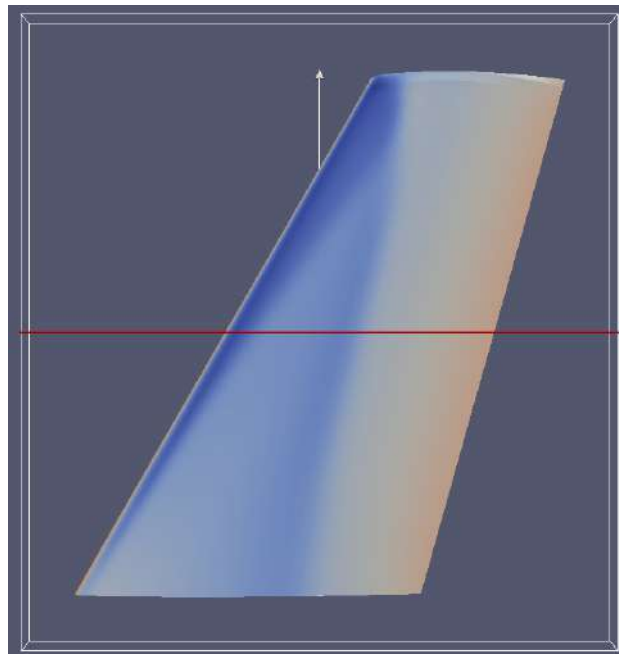


Figure 20.6: Determining the slice to be taken (red line shows the location where a cross-sectional slice will be taken from the wing).

After clicking on the **Apply**, you will generate a slice of the wing span at the specified y-coordinate, as shown in Figure 20.7. To plot C_p along this slice the chord line should first be made non-dimensional. To do this, you can take the following steps:

- 1) Click on the upper left icon, as shown in Figure 20.8. This will create another set of viewing options.
- 2) By click on the *Spreadsheet View*, as shown in Figure 20.9, it will generate a spreadsheet view of the data contained in that slice.
- 3) As seen from Figure 20.10, we want to pan to the right where the data set of **Points** are located. You can vary the display **Precision** when looking up the maximum and minimum

values as well.

- 4) Double click on the **Points** to sort them by descending or ascending order.

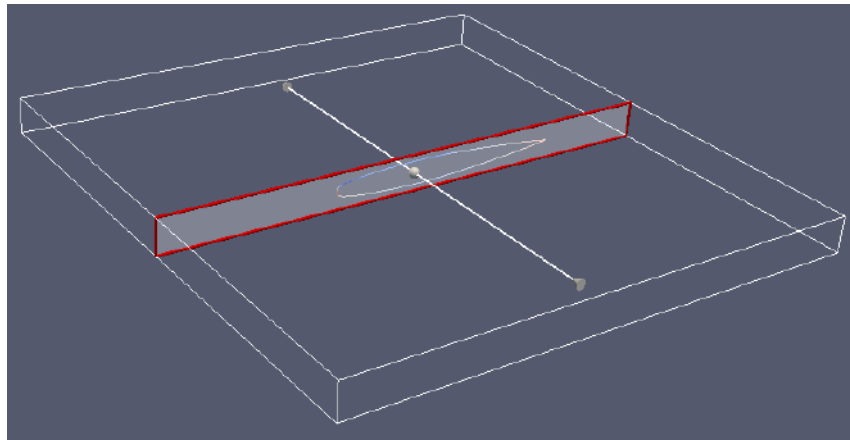


Figure 20.7: Slice to be taken from the wing.

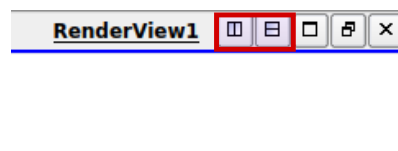


Figure 20.8: How to make alternative viewing options.

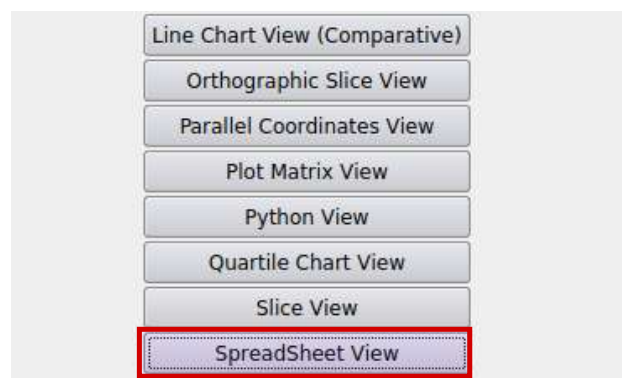


Figure 20.9: How to make spreadsheet view.

We want to get the maximum and minimum values in the array to determine the local chord length on the wing. As shown in Figure 20.10a and Figure 20.10b, the maximum and minimum values for this slice are 0.97631m and 0.349704m, respectively. These two values correspond to x -coordinate for the trailing edge and the leading edge of the chosen wing section, respectively. As you can see in Figure 20.10, the y -coordinate is constant for all values, and hence the slice was taken perpendicular to the y -coordinate.

- 5) Click on *Slice1* in the **Pipeline Browser**, and click on the **Calculator** icon (Figure 20.11).
- 6) We want to create a new array field called *Normalized chord*. We will then plot the pressure coefficient (C_p) along the local normalized chord length. Therefore, as shown in to Figure 20.12, type *Normalized chord* in the **Result Array Name** text-box.

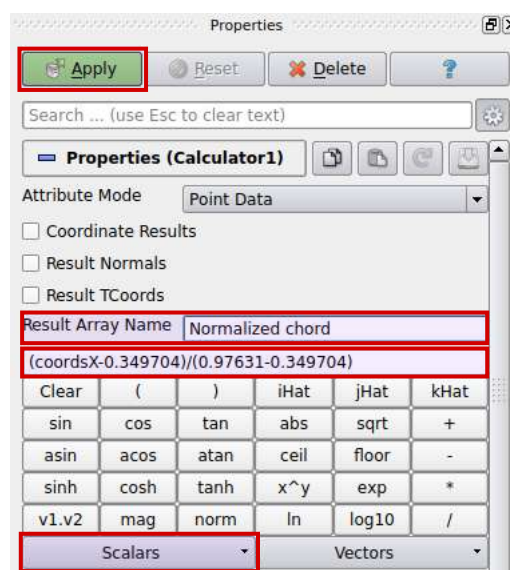
Showing		Attribute: Point Data		Precision: 6			
	Nu_Tilde	Points			Pressure	Pressure_Coefficient	
0	1e-10	0.97631	0.608182	0	1.11537	0.203708	7.72779e-06
1	1e-10	0.967565	0.608182	-0.00119019	1.10181	0.176579	0.000567744
2	1e-10	0.958595	0.608182	-0.00238362	1.0906	0.154172	0.000572503
3	1e-10	0.949206	0.608182	-0.00361078	1.07781	0.128594	0.000613549
4	1e-10	0.939236	0.608182	-0.00489491	1.0652	0.10337	0.000661761

(a) Maximum value for x -component of the **Points**.

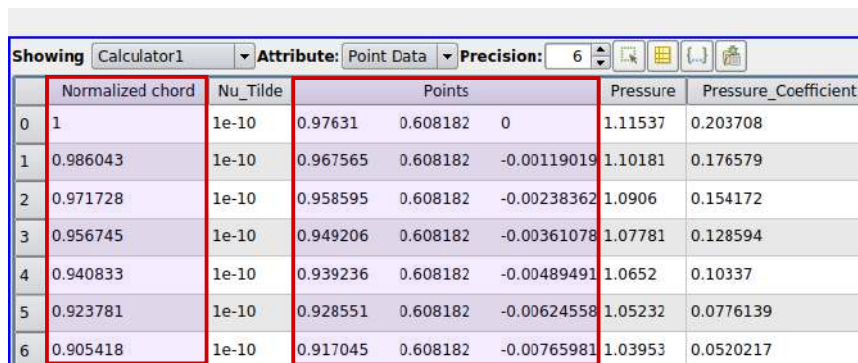
Showing		Attribute: Point Data		Precision: 6			
	Nu_Tilde	Points			Pressure	Pressure_Coefficient	
0	1e-10	0.349704	0.608182	0	1.40143	0.775827	0.
1	1e-10	0.350216	0.608182	-0.00312392	1.43091	0.834781	0.
2	1e-10	0.350216	0.608182	0.00312392	1.27845	0.529878	0.
3	1e-10	0.351728	0.608182	-0.00620189	1.37389	0.720743	0.
4	1e-10	0.351728	0.608182	0.00620189	1.10145	0.175867	0.

(b) Minimum value for x -component of the **Points**.Figure 20.10: How to find maximum/minimum of the **Points**.

- 7) Additionally, type the following equation in the equation-box to normalize the local chord length at that chosen station along the wing span. Please note that the variable *coordsX* can be chosen from the **Scalars** drop-down menu.
- 8) Click on **Apply** to proceed the next step.

Figure 20.11: How to use **Calculator**.Figure 20.12: How to make new variable using **Calculator**.

After taking these steps, the *Normalized chord* is made and added to the list of variables that are available for plotting. You can see *Normalized chord* in *Spreadsheet View*, as shown in Figure 20.13.



	Normalized chord	Nu_Tilde	Points			Pressure	Pressure_Coefficient
0	1	1e-10	0.97631	0.608182	0	1.11537	0.203708
1	0.986043	1e-10	0.967565	0.608182	-0.00119019	1.10181	0.176579
2	0.971728	1e-10	0.958595	0.608182	-0.00238362	1.0906	0.154172
3	0.956745	1e-10	0.949206	0.608182	-0.00361078	1.07781	0.128594
4	0.940833	1e-10	0.939236	0.608182	-0.00489491	1.0652	0.10337
5	0.923781	1e-10	0.928551	0.608182	-0.00624558	1.05232	0.0776139
6	0.905418	1e-10	0.917045	0.608182	-0.00765981	1.03953	0.0520217

Figure 20.13: Normalized coordinate in *Spreadsheet View*.

In order to plot the pressure coefficient along the non-dimensional length of the wing, you can take the following steps:

- 1) Activate *Calculator1* by clicking on it in **Pipeline Browser**.
- 2) Go to **Filter** → **Data Analysis** → **Plot Data** (Figure 20.14). Consequently, *PlotData1* is created and added to the **Pipeline Browser** (Figure 20.15).

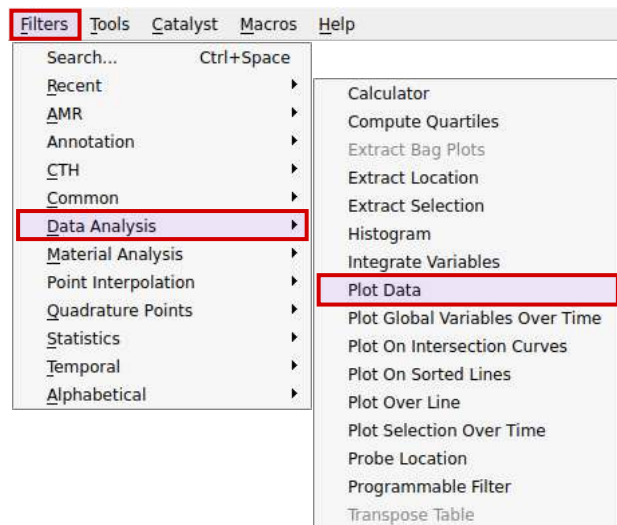


Figure 20.14: How to plot data versus a variable list.

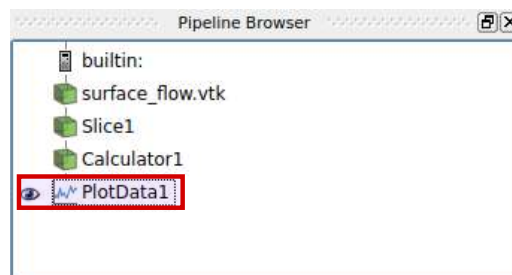


Figure 20.15: *PlotData1* in **Pipeline Browser**.

- 3) In the **Properties** tab, click on the **Display** panel (Figure 20.16).
- 4) Under the **X Axis Parameters** uncheck **Use Index For XAxis**, and then choose *Normalized chord* from the **X Array Name** drop-down menu.
- 5) Unselect everything in the **Series Parameters** fields except for the *Pressure_Coefficient*.
- 6) At the bottom of the **Series Parameters** field, select *Diamond* for the **Marker Style**. After taking these steps, a plot for pressure coefficient should be generated, similar to Figure 20.17.

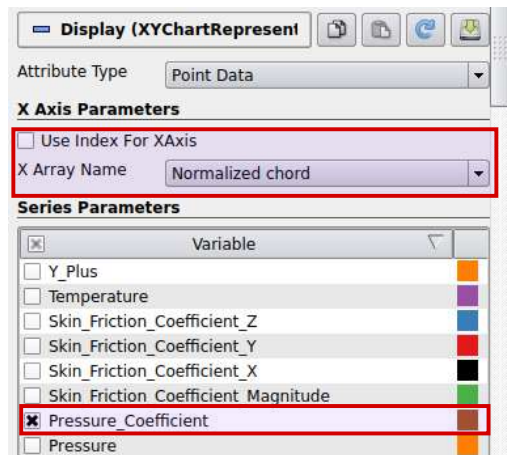


Figure 20.16: How to plot pressure coefficient versus *Normalized chord*.

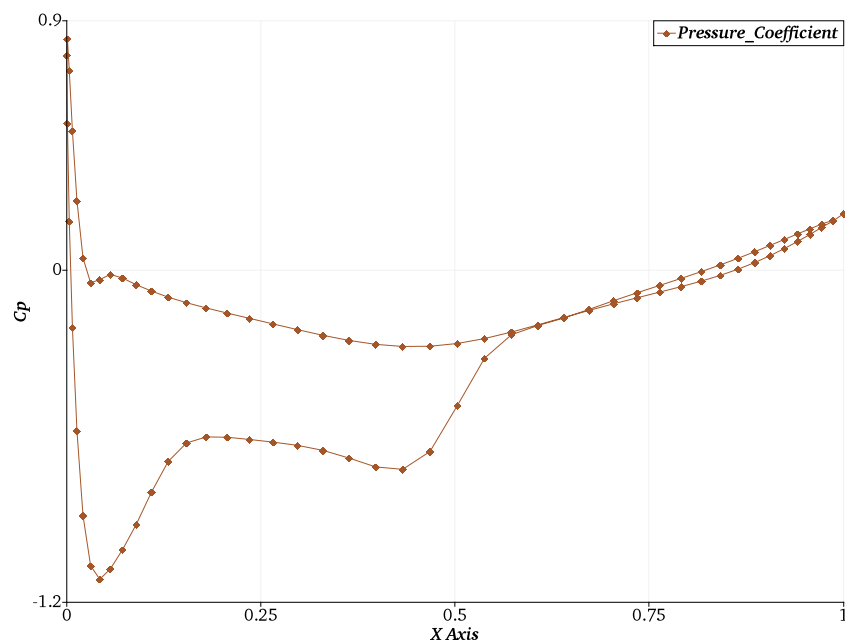


Figure 20.17: Pressure coefficient on the surface of the airfoil versus *Normalized chord*.

Similar to the Chapter 16, we want to have the C_p on the suction surface to be on top of the pressure side. To do this, please take the following steps:

- 7) Go to the **Properties** tab and select the **View** panel.
- 8) As shown in Figure 20.18, under **Left Axis Range** you can click on the check-box beside the **Left Axis Use Custom Range** option, and then switch the values for **Left Axis Range**.
- 9) To rearrange numbers along the left axis, you can click on the check-box beside **Left Axis Use Custom Labels**, and provide different numbers based on your visual preference. Eventually,

the plot you get should be similar to Figure 20.19.

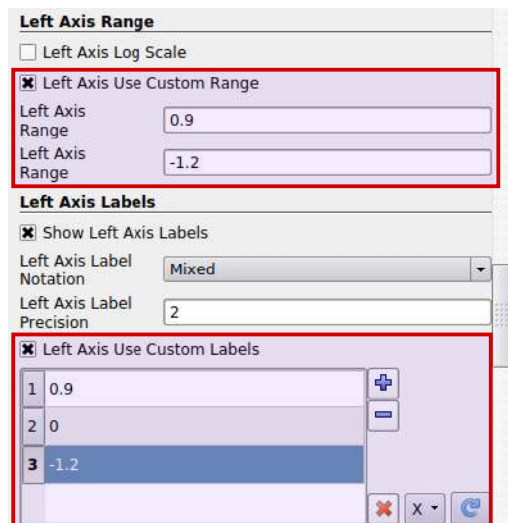


Figure 20.18: Settings for changing the range of variables in left axis of 2D plot.

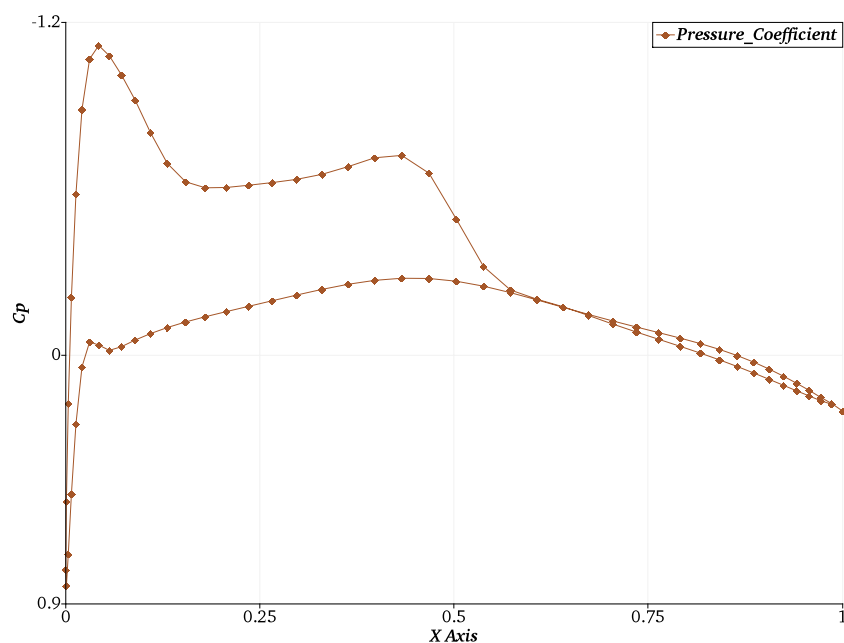


Figure 20.19: Final plot for the pressure coefficient on the surface of the airfoil versus *Normalized chord*.

20.0.4 Visualize Pressure Coefficient (Alternative Method) Exporting .csv file at Different Stations

If you would like to try plotting in other software (i.e. Microsoft Excel), you can collect data for each station (or slice) and then export to .csv file. To get started, you can take the following steps:

- 1) Activate `surface_flow.vtk` by clicking on it in **Pipeline Browser**.
- 2) Go to **Filter** → **Data Analysis** → **Plot On Intersection Curves** (Figure 20.20).
- 3) According to Figure 20.21, under **Plane Parameters** from **Properties** panel, click on the *Y Normal*. Please note that the span of wing is $b = 1.2\text{m}$.

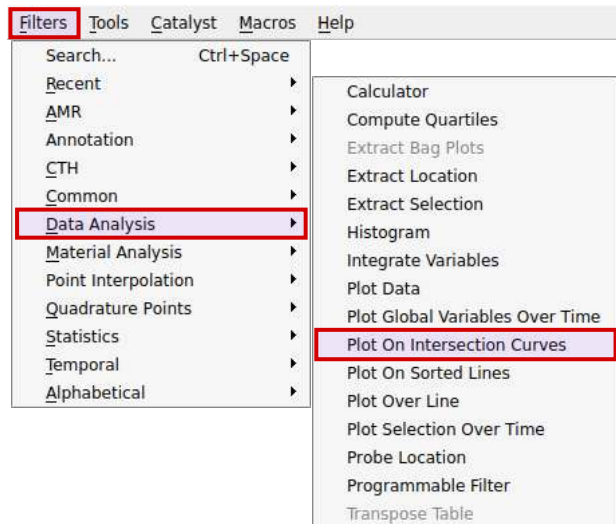


Figure 20.20: How to extract data from an intersection of the surface curve.

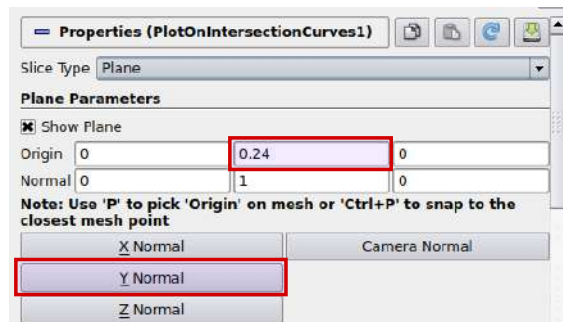


Figure 20.21: Settings for specifying the curve intersection on the wing.

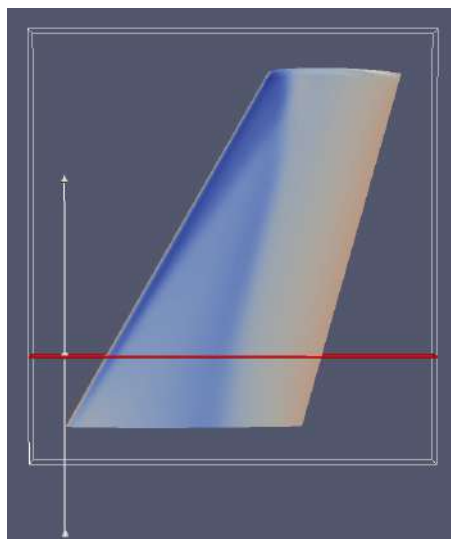


Figure 20.22: Location of intersection (red line shows the locaiton where the intersection of wing curve and normal plate to it is determined).

- 4) To get the C_p curve in station at $y/b = 0.2$, for the **Origin** coordinate, keep the default values of the x -coordinate and z -coordinate; set the y -coordinate to 0.24. This value is obtained by

multiplying the wing span of 1.2 by y/b of 0.2.

- 5) You should see the plane to be located at $y/b = 0.2$ (Figure 20.22), and then, click on the **Apply**.
- 6) You will see *PlotOnIntersectionCurves1* appears as a subset of *surface_flow.vtk* in **Pipeline Browser**.
- 7) In order to export .csv file for $y/b = 0.2$, activate *PlotOnIntersectionCurves1* by clicking on it in the **Pipeline browser**.
- 8) Go to **File** → **Export Scene...**, and save the .csv file.
- 9) To obtain the .csv files for other stations, activate again *PlotOnIntersectionCurves1* in the **Pipeline Browser**, and in the **Properties** panel, change the y -coordinate according to the desired y/b value. For example, to obtain the right location for station with $y/b = 0.44$, multiply this value with 1.2 to obtain the y -coordinate of 0.528. Then, click on the **Apply**, and proceed export procedure as before.
- 10) In the next step, launch Microsoft Excel.
- 11) Open the .csv file you exported earlier in step (6).
- 12) To create the C_p plots, first we need to create a new variable x/c , which is the normalized x -coordinate with respect to the chord length at that station. With the *S Columns* as x -coordinate, enter the following equation in the equation-box in the toolbar to get x/c values:

$$= (S2 - MIN(\$S\$2 : \$S\$90)) / (MAX(\$S\$2 : \$S\$90) - MIN(\$S\$2 : \$S\$90)) \quad (20.1)$$

- 13) For other .csv files, edit the column variable and range accordingly. You can then plot the C_p curve for each station.
- 14) The .csv file does not keep the equations but only the values at each cell. Therefore, to keep the equations, you can save the file as .xlsx.

Questions

1. Plot and comment on the mesh and compare it with the one used for the inviscid ONERA M6. To view the mesh use both the *flow.vtk* and *surface_flow.vtk* files to view the mesh structure from different directions.
2. Run the Onera M6 with Spalart-Allmaras the (SA) turbulence model as described in the tutorial documentation. Plot the pressure coefficient contour and contour lines of the wing surface.
3. Perform Q.2 again but with $k - \omega$ SST turbulence model.
4. Plot the pressure coefficient vs x/c at the stations $y/b = 0.2, 0.44, 0.65, 0.8, 0.9, 0.95, 0.99$ for SA and $k - \omega$ SST with the experimental data provided at the start of this tutorial for each of these stations [19].
5. Compare the C_L and C_D values of the SA and $k - \omega$ SST model; compare as well with the C_L and C_D values obtained from the inviscid case (multigrid results). Comment on possible sources of discrepancies relative to the experimental data.

21. Mesh Generation Using Gmsh

Required Files



Use the following links to download the same version of Gmsh for Windows ([click here](#)) or Mac ([click here](#)).



Use the following links to download the python file for this tutorial ([click here](#)).

Problem Description

In this tutorial, we are going to explain how to generate a mesh for inviscid flow around an airfoil.

Airfoil Coordinate Preparation

To generate the mesh, we first need access to the coordinates defining the desired airfoil's geometry. These are typically obtained from reference material, or from the shape parameterization of standard airfoil families, such as the NACA series. In this tutorial, we will use AirfoilTools.com (Figure 21.1) to obtain a set of coordinates for our airfoil. For example, we can select either the **NACA 4 digit generator** or **NACA 5 digit generator** on this website. Here, we will demonstrate mesh generation for a NACA2312 using the **NACA 4 digit generator** (Figure 21.2). To get started, you can take the following steps:

- 1) Go to AirfoilTools.com using a web browser.
- 2) From the main page, click on the **NACA 4 digit generator**, on the left-hand side of the web page.



Figure 21.1: The www.airfoiltools.com website.

- 3) According to the NACA2312 specification, the parameters defining the **Max Camber**, **Max camber position**, and **Thickness** are 2, 30 and 12, respectively.
- 4) Select the **Number of points** to be 50. Please note that it can be increased if more precision is required.
- 5) Make sure you activate the check-box beside **Cosine spacing** to choose the **Close Trailing edge** option. The **Cosine spacing** increases the number of points (or decreases the distance between points) around the leading edge and trailing edges of the airfoil, which have the greatest curvature. Additionally, **Close Trailing edge** forces a sharp trailing edge on the airfoil.

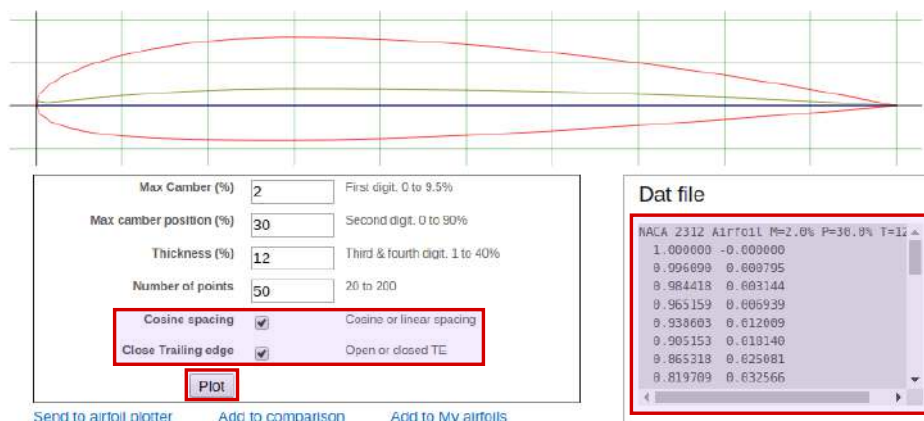


Figure 21.2: Settings for obtaining a NACA 4 digit series airfoil coordinates.

- 6) Click on the **Plot**, and copy the coordinate data from the **Data file**.
- 7) Create a new file and open it with any standard text editor (i.e. Notepad, Text Editor, etc.)
- 8) Copy the coordinates into it and finally, go to **File** → **Save as** and choose a suitable filename, such as `naca2312.dat`.

Converting Coordinates to Gmsh format

Gmsh uses a native `.geo` file format, which is a readable list of instructions to build the airfoil geometry, and then mesh it. Points in Gmsh can be defined in the `.geo` file as:

```
Point(1)={1.00000000, 0.00000000, 0.00000000, 1.000};
```

where, the number in `Point(1)` shows the index of the corresponding point, and the first three numbers in brackets indicate the point coordinates in space (its x , y , and z components). Additionally, the last number is the scale of the mesh distribution around the corresponding point, which is 1 by default. Keep in mind that the coordinates saved in the `naca2312.dat` include only the x and y coordinates, whereas Gmsh also requires the z coordinate. Hence, for a two-dimensional case we must also add zero for the z component to the `naca2312.dat` file. Since it can be time-consuming to input all of the airfoil geometry points into Gmsh, we have included with this tutorial a small Python script that reads the `naca2312.dat` file, converts the coordinates into Gmsh format, and then exports a new `.geo` file with all of the points connected using a spline. To use this converter, copy the `gmshconverter.py` script file into the folder you saved `naca2312.dat`, and then open a terminal window and enter the following commands:

Windows	\$ cd "where you saved naca2312.dat and gmshconverter.py" \$ python3 gmshconverter.py naca2312.dat naca2312.geo
Mac	\$ cd "where you saved naca2312.dat and gmshconverter.py" \$ python3 gmshconverter.py naca2312.dat naca2312.geo

After completing these steps, the `.geo` file is saved in the same directory and is ready to be loaded into Gmsh.

Mesh Design Using Gmsh

In this section, we will show how to use Gmsh. In order to get started, you can take the following steps:

- 1) Launch Gmsh.
- 2) Go to **File** → **Open**, and select the `naca2312.geo` file. As shown in Fig21.3, once you have loaded the `naca2312.geo` file you will see the points defining the airfoil surface in the display window.
- 3) To make a surface for the airfoil we can use a spline to define a smooth curve. On the right-hand side of the Gmsh display window, go to **Modules** → **Geometry** → **Elementary entities** → **Add** → **Spline** (Figure 21.4).

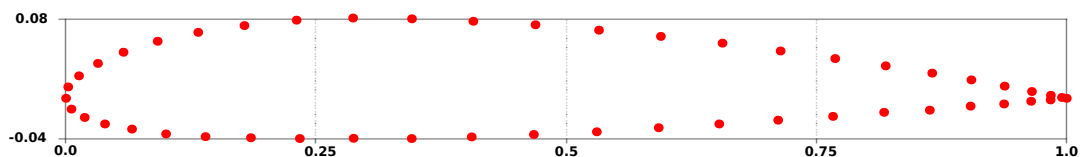


Figure 21.3: Points defining NACA2312 airfoil.

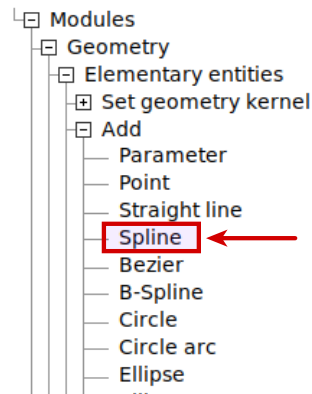


Figure 21.4: How to add spline to design platform.

- 4) Start to click on points from the trailing edge (1,0,0) toward the leading edge (0,0,0) for the upper side of the airfoil.
- 5) When you reach (0,0,0) press the ϵ key on your keyboard to apply your command (or design).
- 6) Then, start again from (1,0,0) toward (0,0,0) for the lower side of the airfoil, and again when you reach (0,0,0), press the ϵ button.
- 7) It is worth pointing out that by pressing the ϵ button, Gmsh applies the selected command on the geometric entities you have selected. Additionally, you can quit the current command by pressing the η button.
- 8) Once you are done making the upper and lower surfaces of the airfoil, press η to exit from the spline command. As shown in Figure 21.5, splines for both the upper and lower surfaces of the airfoil are now added to the geometry.

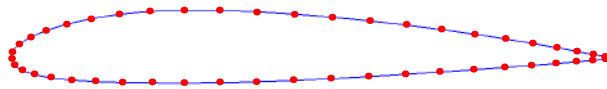


Figure 21.5: Splines added to define a smooth surface for the NACA2312 airfoil.

- 9) In order to define the outer boundaries for the computational domain, as shown in Figure 21.6, go to **Modules** → **Geometry** → **Elementary entities** → **Add** → **Point**.

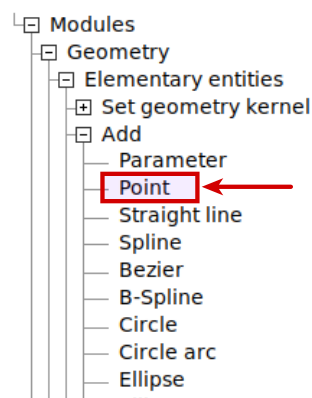


Figure 21.6: How to add points to define the outer domain boundaries.

- 10) Another display window similar to Figure 21.7 will appear that you can use to define the coordinates for each boundary point.
- 11) Click on the **Apply** button to add them in order.
- 12) Give four points with following coordinates: (10,-10,0), (10,10,0), (-10,-10,0), and (-10,10,0).
- 13) Next, press the \square button on your keyboard to terminate the new point command. As seen from Figure 21.8, four points are added in the display window.

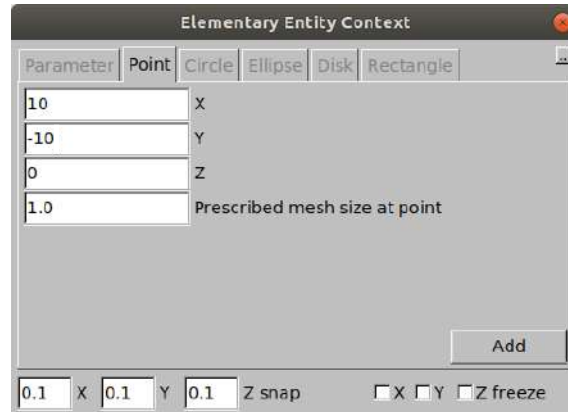


Figure 21.7: Adding points to the domain.

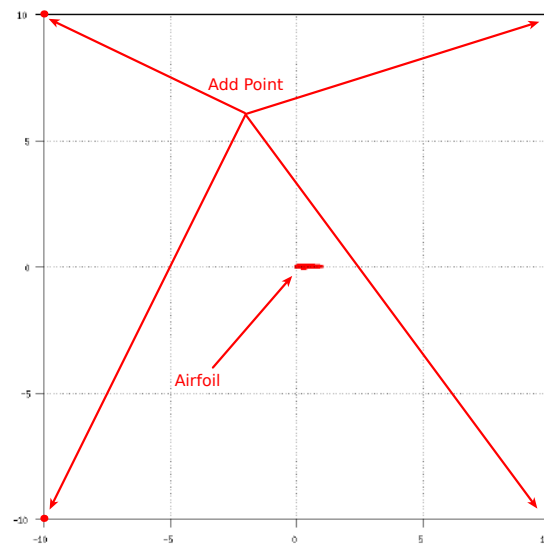


Figure 21.8: Final points defining the outer edges of the domain.

- 14) The next step is to connect these four points using lines. To do this, as shown in Figure 21.9, go to **Modules** → **Geometry** → **Elementary entities** → **Add** → **Straight line**.
- 15) Connect the points by clicking on them in order.
- 16) Once they are connected, four straight blue lines showing the connection between each pair of points should be visible (Figure 21.10).
- 17) Now press the \square button on your keyboard to end the command.
- 18) The next step is to specify the boundaries of the domain/geometry with respect to the lines/splines, and define a unified computational domain. As shown in Figure 21.11, go to **Modules** → **Geometry** → **Elementary entities** → **Add** → **Plane Surface**.

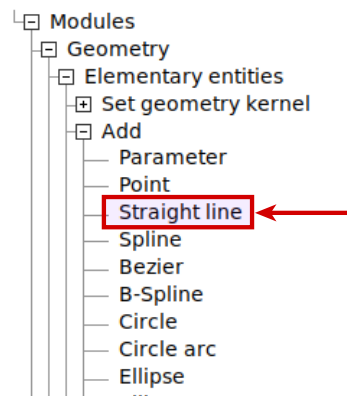


Figure 21.9: How to add straight lines to the domain.

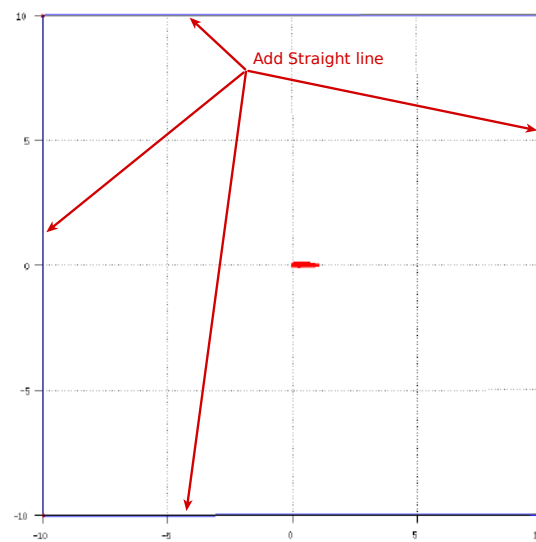


Figure 21.10: Straight lines added to define the outer boundaries of the domain.

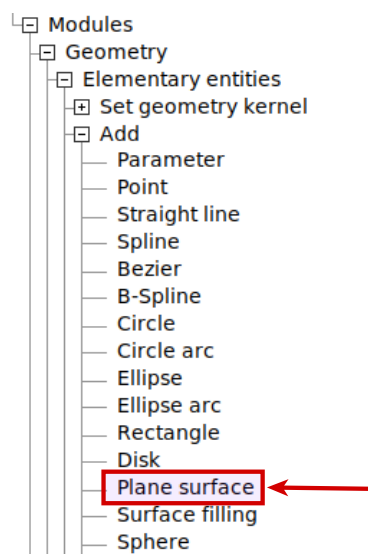


Figure 21.11: How to add a plane surface to the domain.

- 19) Click on all lines and splines, and then, press ϵ button on your keyboard. As shown in Figure 21.12, dashed grey lines will now appear, which defines a computational surface that defines the region containing fluid flow.
- 20) Press the η button on your keyboard to end the command.

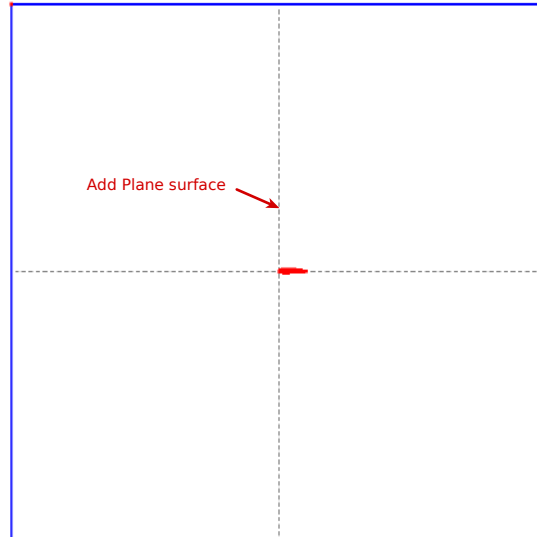


Figure 21.12: A new surface that defines the computational domain.

- 21) After defining the computational domain in Gmsh, we also need to specify the boundary conditions tags that will be read by SU2 (or any other compatible CFD software). As shown in Figure 21.13, go to **Modules** → **Geometry** → **Physical groups** → **Add** → **Line**.

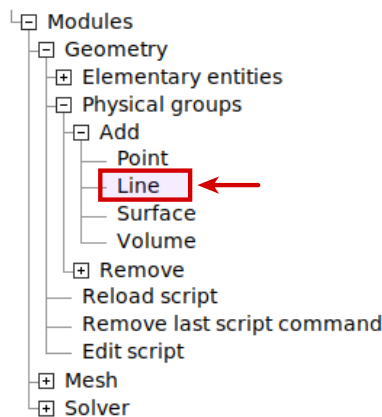


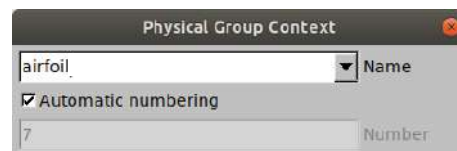
Figure 21.13: How to define boundary type in design platform.

- 23) To define the outer boundary condition, as shown in Figure 21.14a, type *farfield*, and then select the four straight lines in the display window by clicking on them. This step specifies the boundary condition name of these four lines as *farfield*, and later SU2 will read in this boundary tag *farfield*.
- 24) Press the ϵ button on your keyboard.
- 25) As shown in Figure 21.14b, do the same thing for the airfoil surface, but giving it the name *airfoil*.
- 26) to specify the fluid containing region of the domain go to **Modules** → **Geometry** → **Physical groups** → **Add** → **Surface** (Figure 21.15).

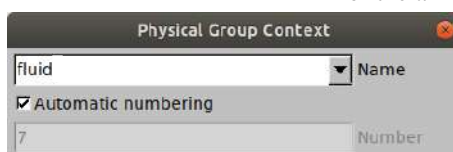
- 27) As shown in Figure 21.14c, type *fluid* in the text-box, and then select surface of the domain in the display window by clicking on one of the dash lines (which indicate the surface of the domain).
- 28) Press the ϵ button to apply this tag, and then the α to exit the command.



(a) Define *farfield* boundary for outer boundaries



(b) Defining *airfoil* boundary for the surface of the airfoil



(c) Defining *fluid* for the inside of the computational domain.

Figure 21.14: How to define a new range for the contour lines.

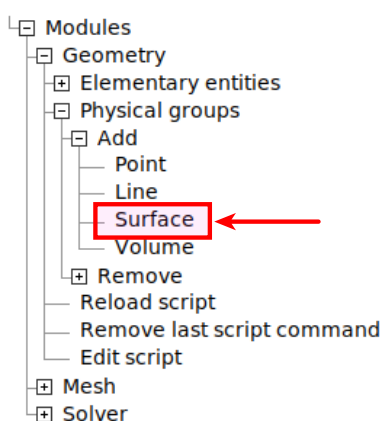


Figure 21.15: Defining different boundary type.

In order to add elements to the domain, we start by specifying the number of points to use on each line/spline, and then generate an unstructured mesh over the entire domain. To get started, you can take the following steps:

- 1) Go to **Modules** → **Mesh** → **Define** → **Transfinite** → **Line** (Figure 21.16).
- 2) As shown in Figure 21.17a, choose 100 grid points in the **Number of points** option.
- 3) select *Bump* from the **Type** drop-down menu, and set 0.3 for this **Parameter**. In this step, we selected 100 grid points for each side of the airfoil, and by selecting *Bump* with 0.3 as parameter set, we distributed the nodes along each spline with a concentration near the leading and trailing edges of the airfoil. You can change the node clustering by choosing different values for the bump **Parameter**.
- 4) Click on both splines the form the upper and lower surfaces of the airfoil and press the ϵ button on your keyboard.
- 5) According to Figure 21.17b, set the **Number of points** option to 20.
- 6) To have similar distributions of the grids on each line of the outer boundaries, select *Progres-*

sion from **Type** drop-down menu with value of 1 for the **Parameter**.

- 7) Click on all four straight lines at the outer boundary of the domain, and then press the **e** key on your keyboard.
- 8) At the end, press the **q** key to end the command.
- 9) The next step is to finally generate the mesh for the 2D domain. To do this, as shown in Figure 21.18, go to **Modules** → **Mesh** → **2D**. Now you can select **2D**, and it will automatically generate an unstructured mesh for the computational domain.

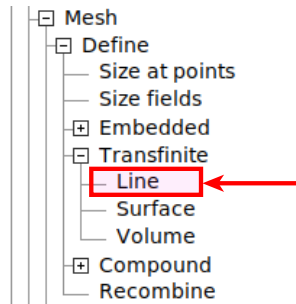
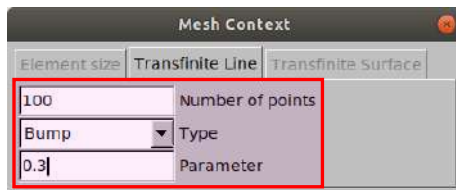
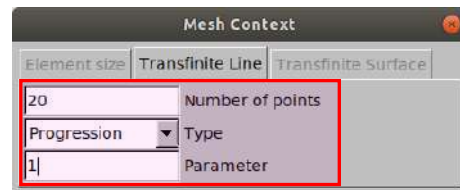


Figure 21.16: How to specify the number of nodes on the boundaries.



(a) Grid points for the surface of the airfoil.



(b) Grid points for the outer boundaries.

Figure 21.17: Settings for defining different type of grid points.

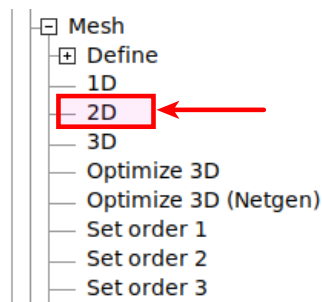
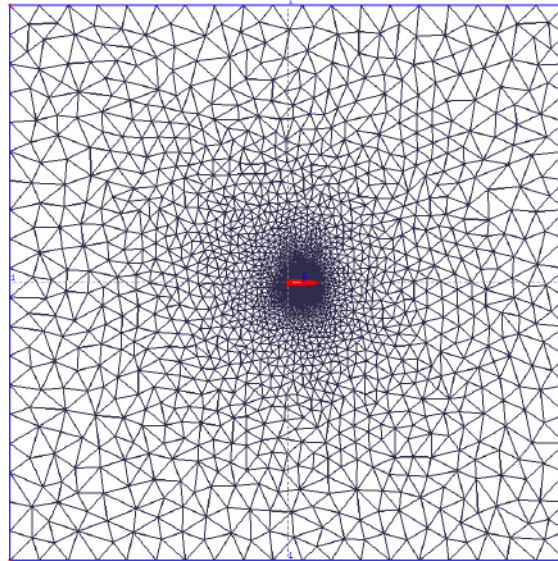


Figure 21.18: How to generate 2D mesh.

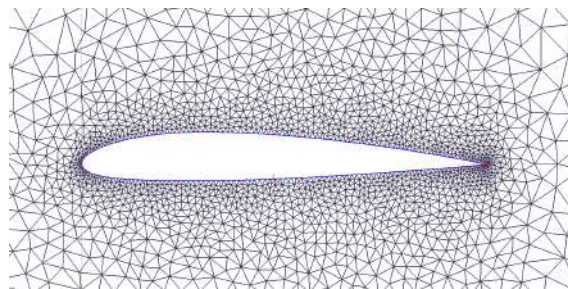
The mesh you get should be similar to that shown in Figure 21.19. As you can see from Figure 21.19a, the mesh is coarse around the outer boundaries, while it is significantly finer around the airfoil (Figure 21.19b), especially near the leading edge and trailing edges of the airfoil geometry since we used the bump distribution on these lines.

Once the mesh generation is completed, we need to save it in a format that can be read by SU2. Fortunately, Gmsh can save directly in the native SU2 format generating a `.su2` mesh file. To do this, go to **File** → **Export**. Next, according to Figure 21.20, select **Mesh-SU2 (* .su2)** from the **Format** drop-down menu, and in the **Filename** address bar, type `naca2312 . su2` to create a new `.su2` mesh file. Then, click on **OK** to proceed to the next step. A display window appears that

asks for various **SU2 Options**. You should ignore this option for now by clicking on **OK** to simply write the mesh file to disk. You should now see the `naca2312.su2` mesh has been generated and saved.



(a) Full view of computational domain with unstructured mesh.



(b) Zoom-in view of the computational domain.

Figure 21.19: Mesh generated for computational domain.

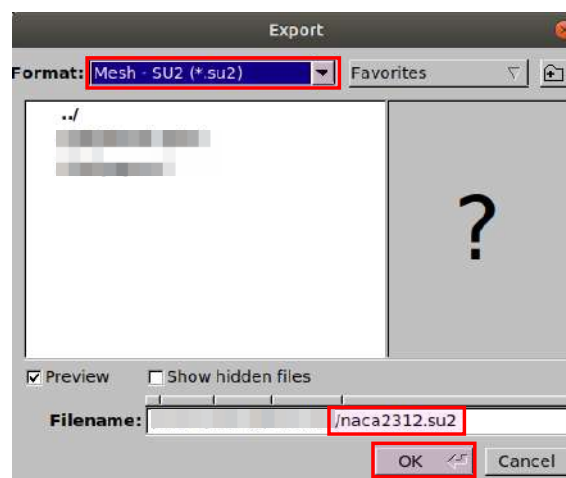


Figure 21.20: How to export the mesh in SU2 format.

Run Tutorial In Chapter16 Using New Mesh

To demonstrate how to use this mesh, we will repeat the tutorial in Section (16) but using this new NACA 2312 mesh. To choose the new mesh, and to change the names of boundary conditions, you can open the SU2 configuration (.cfg) file. For example, open the `inv_NACA0012.cfg` file and search for INPUT/OUTPUT INFORMATION. According to Figure 21.21, in MESH_FILENAME under INPUT/OUTPUT INFORMATION, you can change the filename of mesh you are going to use, in this case using `naca2312.su2`.

```
% ----- INPUT/OUTPUT INFORMATION ----- %
% Mesh input file
MESH_FILENAME= mesh_NACA0012_inv.su2
%
% Mesh input file format (SU2, CGNS, NETCDF_ASCII)
MESH_FORMAT= SU2
%
```

Figure 21.21: Changing the mesh in the SU2 configuration file.

Additionally, if you would like to use a different name for the physical boundaries in the Gmsh, you should change the boundary names in the configuration file as well. To do this, search for BOUNDARY CONDITION DEFINITION in `inv_NACA0012.cfg`. As shown in Figure 21.22, you can change the names of your boundaries to be the same as those used in Gmsh when you defined your Physical Lines/Curves. In this case, `MARKER_EULER` and `MARKER_FAR` correspond to the wall boundary for the upper and lower surfaces of the airfoil, and the far-field boundary, respectively. After having these changes, you can continue with the rest of the steps, similar to Section (16).

```
% ----- BOUNDARY CONDITION DEFINITION ----- %
%
% Marker of the Euler boundary (NONE = no marker)
MARKER_EULER= ( airfoil )
%
% Marker of the far field (NONE = no marker)
MARKER_FAR= ( farfield )
```

Figure 21.22: Boundary conditions settings in the SU2 configuration file.

Questions

- Run the provided default NACA2312 test case at the provided $Ma = 0.8$.
 - Plot and comment on the mesh.
 - Plot and comment on the pressure contours around the airfoil.
 - Plot and comment on the pressure coefficient on the surface of the airfoil.
- Re-run the default NACA2312 case but change the Mach number to $Ma = 0.3$ and run several simulations using $\alpha = 0, 2, 4, 6, 8, 10, 12, 14, 16$ degrees.
 - Plots C_l vs α alongside the provided experimental data[10] in the test case folder.
 - Repeat 1.b with $Ma = 0.3$ for the $\alpha = 0, 8, 16$ degree cases.
 - Repeat 1.c with for the $\alpha = 0, 8, 16$ degree cases.
- Discuss the differences exist between the results of this tutorial and the results of Chapter16. What is the major difference and what could be the reason for that?

22. Shock Waves

Required Files



Use the following links to download the same version of SU2 for Windows ([click here](#)) or Mac ([click here](#)). The required configuration and mesh files for normal shock detection ([click here](#)). The required configuration and mesh files for oblique shock detection ([click here](#)).



Use the following links to download the same version of Paraview for Windows ([click here](#)) or Mac ([click here](#)).

Problem Description

When the speed of flow is much less than the speed of sound ($Ma < 0.3$), the density of flow remains constant. However, when the speed of flow is higher ($0.3 < Ma < 1$), then the compressibility effect becomes an important matter. For subsonic compressible flows, the entropy remains constant and the flow process is reversible. On the other hand, when the flow speed exceeds the speed of sound ($Ma > 1$), the flow experiences abrupt changes in volume, and hence notable changes in density. In this case, the flow process becomes irreversible, and hence, the entropy increases. This significant and fast change in the density of gas flow is the main reason for the shock formation. Shocks are a small region in the gas where the flow properties change significantly. When shock happens, the static pressure and temperature increases, while the Mach number decreases. It is worth mentioning that if the shock waves are perpendicular to the flow direction, then this shock is called a normal shock. Another type of shock, which is not perpendicular to the flow direction is called an oblique shock, which is usually formed in supersonic jet flows.

In this tutorial, we are going to demonstrate how to simulate a supersonic inviscid flow inside of a Converging-Diverging (CD) nozzle. For simplicity, this nozzle is assumed to have a rectangular cross-sectional area with a depth of 1m. Other geometry specifications are set as follows:

- Inlet height = 280mm
- Throat height = 210mm

- Exit height = 336mm

To begin, let us consider different flow regimes inside the CD nozzle. Figure 22.1 shows a schematic of plot of pressure ratio ($\frac{p_b}{p_0}$) versus centerline location, x . The pressure ratio shows the relation between the pressure at the inlet of the nozzle (p_0) and the exit of the nozzle (p_b). As it is shown in Figure 22.1, four different flow regimes can usually happen in a nozzle given different circumstances. In the first regime noticed by curve (a), when the back pressure of the nozzle is less than the pressure of the inlet flow ($\frac{p_b}{p_0} < 1$), then the flow is subsonic since it does not reach $Ma = 1$ at all. In this case, the gas flow accelerates during the converging part of nozzle, and it decelerates in the diverging part, exiting the nozzle as a subsonic flow. In the curve (b), the flow regime is called sonic flow. In this regime, the gas velocity increases in the converging part and it reaches $Ma = 1$ at the throat. However, the gas velocity decreases in the diverging part and it becomes subsonic at the exit of the nozzle. This flow regime is referred to as choked flow. It means any further reduction in the back pressure of the nozzle does not change the mass flow in the nozzle. Furthermore, curve (c) shows the flow accelerates in the converging part of the nozzle and it reaches $Ma = 1$ at the throat. Also, the gas flow velocity increases continually as the cross-sectional area gets bigger in the diverging part ($Ma > 1$). However, a normal shock happens in the downstream of the throat in the diverging part. In this case, it is expected to have a subsonic flow at the exit of the nozzle. With further reduction in back pressure, when the pressure difference between the inlet and exit of the nozzle is large enough, the flow becomes supersonic in the diverging part of the nozzle and is exhausted as a supersonic flow without any normal shock occurred inside of the nozzle. However, the oblique shock is expected at the exit of the nozzle. As mentioned before, this type of shock is not perpendicular to the flow direction, and it usually happens when a supersonic jet discharges in ambient.

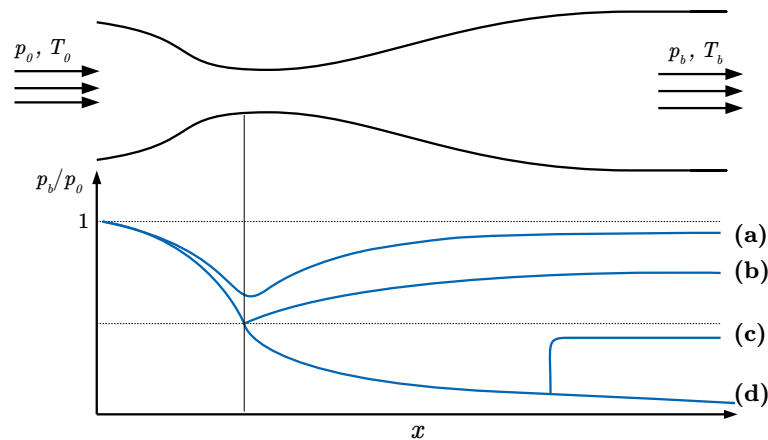


Figure 22.1: Schematic representation of the pressure changes inside the nozzle for different flow regimes.

Usually, the static pressure and temperature at the inlet of the nozzle are given, and based on the operating Mach number, the stagnation pressure and temperature at the inlet of the nozzle are calculated. Considering mathematical formulations is beyond the concept of this tutorial and hence we do not discuss them here. The flow properties for the simulation can be represented as follows:

- Inlet Mach number = 0.5
- Inlet static pressure = 200,000 Pa
- Inlet static temperature = 288 K
- Inlet stagnation pressure = 238,828 Pa
- Inlet stagnation temperature = 308.4 K

- Back (Exit) static pressure = 190,000 Pa

This tutorial has two parts: Flow Solution and Post-processing. In the first part, we will explain how to manage the prerequisite files and settings, and how to run the CFD simulation using SU2. In the second part, we explain how to use Paraview to visualize the data obtained from SU2.

Flow solution

To run this simulation, SU2 needs two files: a configuration file (.cfg) and a mesh file (.su2). Links to the required files and executables are provided at the start of this tutorial. The files include:

1. `inv_normal_shock.cfg` which is a configuration file.
2. `mesh_nozzle.su2` which is a mesh file.

The next step is to copy these two files into the same directory as the SU2 executable. Then, to run the simulation, open a terminal window and enter the following commands:

Windows	\$ cd "where you saved the package" \$ SU2_CFD.exe inv_normal_shock.cfg
Mac	\$ cd "where you saved the package" \$./SU2_CFD inv_normal_shock.cfg

The SU2 solver will commence solving the problem and will print out the residuals at every iteration until the specified convergence criteria is achieved. The computational time for this case is highly dependant on the computer's performance. However, the run time is expected to be about 15 minutes on average. After the calculations are complete, the following output files should have been generated within in the SU2 folder:

- `flow.vtk`: The flow solution on the entire domain.
- `history.vtk`: Convergence history.
- `restart_flow.dat`: Restart file.
- `surface_flow.vtk`: The flow solution on the airfoil surface.

Please keep in mind that every time you run SU2, the output data will be overwritten. Hence, before launching a new simulation you should backup your files in another directory.

Post-Processing

In this section, we explain how to use Paraview to visualize the solution generated by SU2. First of all, install paraview (if not already done) using the links at the start of this tutorial. Once that is complete, perform the following steps to visualize the results:

22.0.1 Load the Solution File:

- 1) Launch Paraview.
- 2) Go to **File** → **Open**, and then select the `flow.vtk` file. On the left side of the Paraview window, you will see the file appears in the **Pipeline Browser** under **builtin**
- 3) Now press the **Apply** button in the **Properties** tab, right under the **Pipeline Browser** heading. After taking these steps, your file is loaded by Paraview and is ready to be visualized (Figure 22.2).

22.0.2 Visualize the Mesh

As shown in Figure 22.3, in order to view the mesh select *Solid Color* with *Wireframe* in the toolbar. Then, you can zoom in to see the mesh employed for this computational domain, as shown in Figure 22.4. As you can see, the mesh in the nozzle is structured, and mesh concentration around

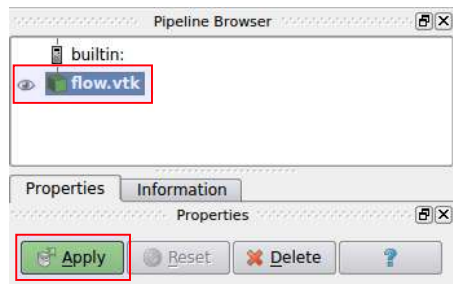


Figure 22.2: Loading the `.vtk` file into the **Pipeline Browser**.

the throat is higher than in other areas since the speed of the flow increases at the throat. Therefore, more cells are needed in this region to detect the flow properties with enough accuracy.



Figure 22.3: How to display the mesh.

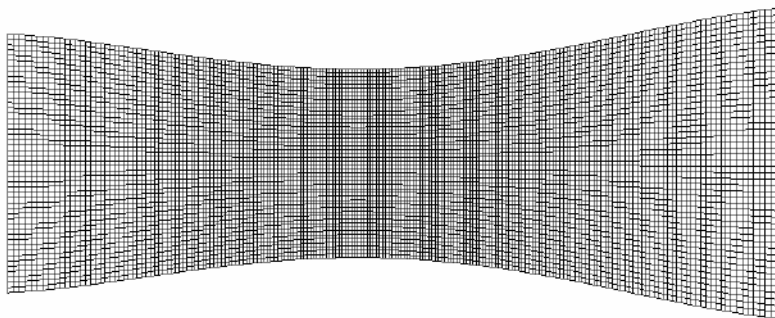


Figure 22.4: The structured mesh for the CD nozzle.

22.0.3 Visualize Pressure, Temperature and Mach Contours

To display pressure contours, you can take the following steps:

- 1) Click on `flow.vtk` in the **Pipeline Browser**, and then click on the **Display** form in the **Properties** tab.
- 2) Under the **Coloring** section, select *Pressure* from the drop-down menu (Figure 22.5). The pressure contours should be like Fig(22.6)
- 3) Repeat the same procedure to get the contours for temperature and Mach number. The contours for the temperature and Mach number should be like Fig(22.7) and Fig(22.8), respectively.

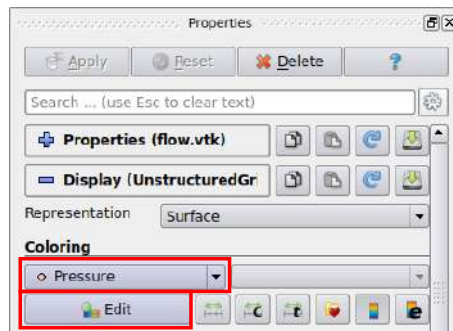


Figure 22.5: Settings for displaying pressure contours.

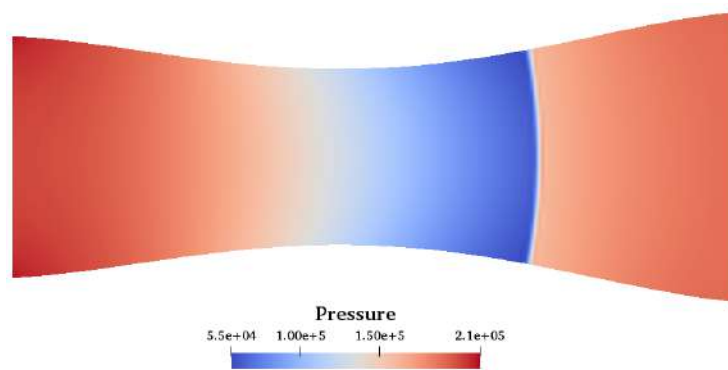


Figure 22.6: Pressure contours for the nozzle.

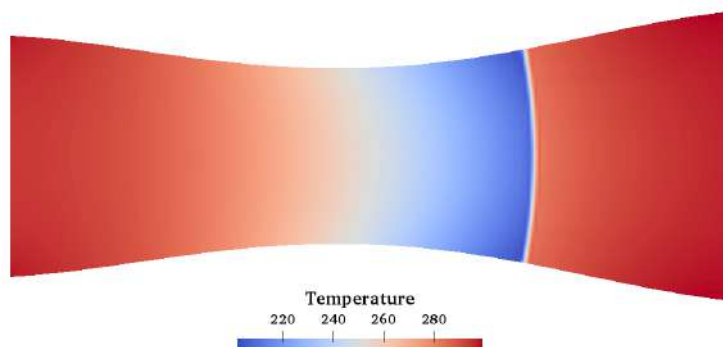


Figure 22.7: Temperature contours for the nozzle.

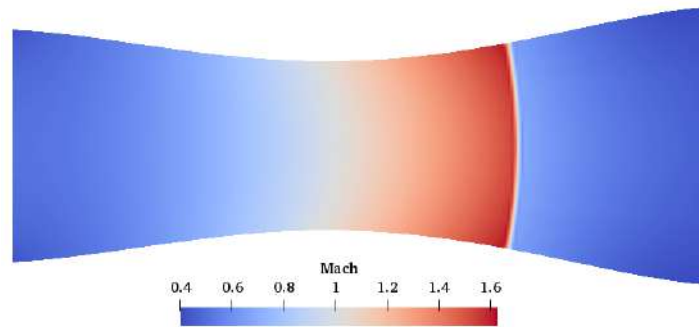


Figure 22.8: Mach contours for the nozzle.

As seen from Figure 22.6, The static pressure decreases even in the diverging part, while after the shock, the pressure increases abruptly. Since the flow process is irreversible, it would not be isentropic anymore. Hence, the total pressure at downstream of shock is always less than the total pressure in upstream. This pressure loss is due to the shock wave. As mentioned before, according to Fig(22.7), the static temperature also increases downstream of the shock wave. But keep in mind that there is no work done in a shock and no heat is produced then. Therefore the total temperature and total enthalpy remain constant. According to Fig(22.8), the Mach number decreases abruptly due to the shock. Roughly speaking, this phenomenon is unfavorable, and to avoid the normal shock in CD nozzles, the pressure difference between the inlet and exit of the nozzle should be increased adequately if supersonic flow is demanded at the exit of the nozzle.

22.0.4 Plotting non-dimensional variables along the centerline

Since static temperature and pressure as well as the speed of gas flow have different dimensions, then plotting all of these three variables in one figure does not fit well. Therefore, they should be non-dimensionalized to have the same scale in the plot. To non-dimensionalize these variables, we can divide the pressure and temperature by their constant values at the inlet of the nozzle. Additionally, for gas velocity, we can use the Mach number since it is already a non-dimensional value. To plot multiple non-dimensional variables along the centerline you can take the following steps:

- 1) Select `flow.vtk` by clicking on it in the **Pipeline Browser**.
- 2) According to Figure 22.9, click on the **Calculator** icon .
- 3) Under **Properties**, click on the **Scalars** and from the drop-down list, select *Temperature* (Figure 22.10).



Figure 22.9: **Calculator** option in Paraview.

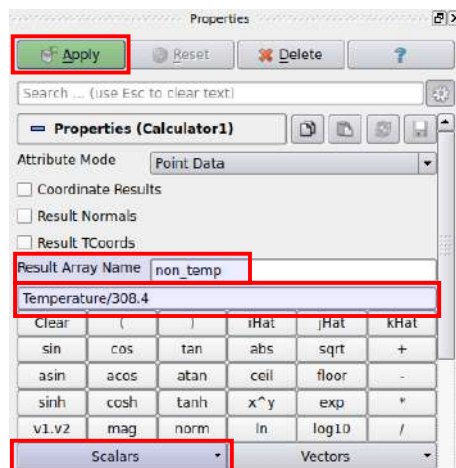


Figure 22.10: Calculator settings.

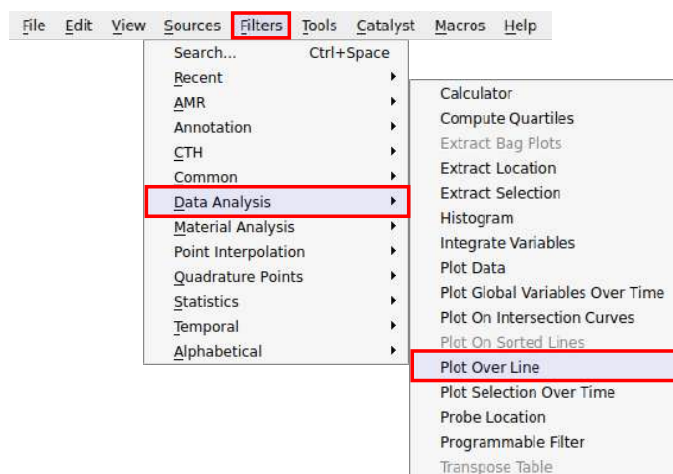


Figure 22.11: How to plot data over an arbitrary line in space.

- 4) Then in the equation-box, divide the *Temperature* by stagnation temperature ($T_0 = 308.4K$).
- 5) Select a name for the non-dimensionalized temperature in **Result Array Name**. Here, we call it *non_temp*. Then, Click on **Apply**.
- 6) Repeat the same procedure for pressure. Please note that you need to divide static pressure by stagnation pressure ($P_0 = 238,828Pa$) and set a name for this variable as *non_pressure*.
- 7) Go to **Filters** → **Data Analysis** → **Plot Over Line** (as shown in Figure 22.11).
- 4) Under **Line Parameters** in **Properties** (as shown in Figure 22.12), select the coordinates for two arbitrary points you want (i.e. **Point1** and **Point2**). In this case, plot the line from **Point1** to **Point2** with (0,0,0) and (0.84,0,0), respectively, and then click **Apply**. A new line plot item will be generated.
- 5) According to Figure 22.13, under the **X Axis Parameters** from **Display**, select *Point_X* from the **X Array Name** drop-down menu. Now, under the **Series Parameters** options toggle the box beside **Variable** to uncheck everything, then select the *Mach*, *non_pressure* and *non_temp* items. The plot of these three variables versus the *x-coordinate* will be displayed in the main window as shown in Figure 22.14.

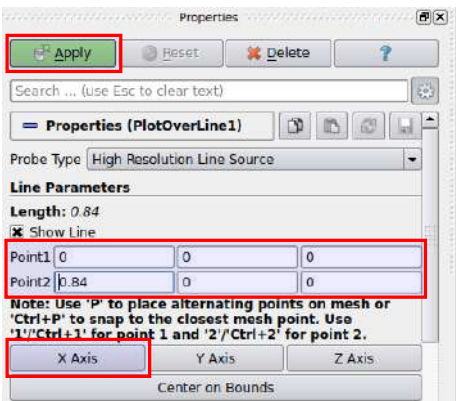


Figure 22.12: Define the coordinates for a line plot in space.

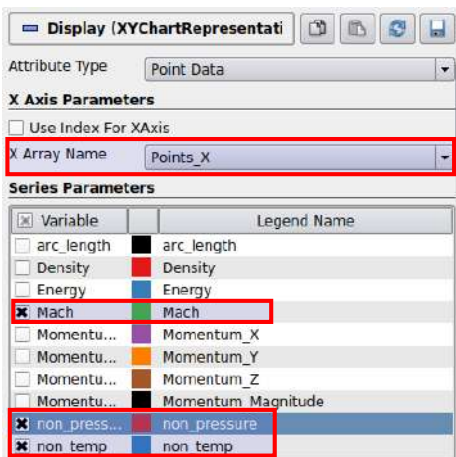


Figure 22.13: How to plot multiple variables over a line.

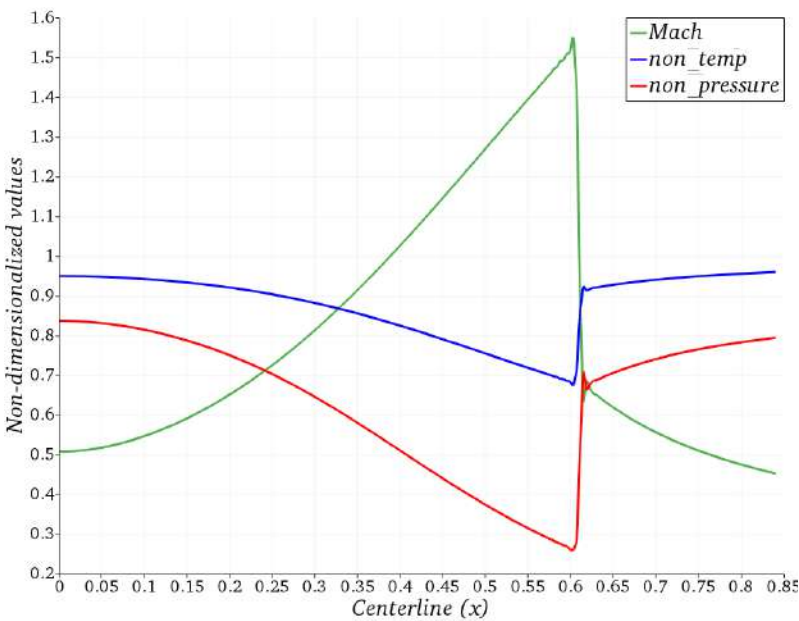


Figure 22.14: Plot for non-dimensionalized variables along the nozzle centerline.

- 4) In order to see a spreadsheet view of the datapoints, you can click the upper right icon as shown in Figure 17.23, then click on the *Spreadsheet View* similar to Figure 17.24.
- 5) Additionally, you can export a .csv file and plot it with any other plotting software. To export the spreadsheet as a .csv file, go to **File** → **Export**, and then **Save as .csv**.

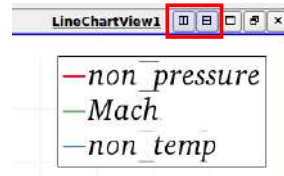


Figure 22.15: How to make spreadsheet view in a new window.

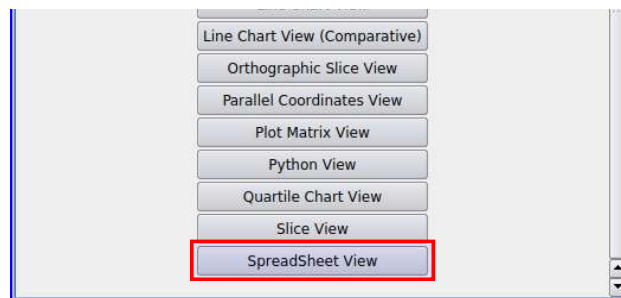


Figure 22.16: Selecting the *Spreadsheet View* among the different available views.

Questions

1. Run the default case as provided which uses JST Riemann solver and detect flow regime represented by curve (c) in Figure 22.1.
 - (a) Create colored contours of Pressure, Temperature, and Mach number.
 - (b) Plot non-dimensionalized Pressure, Temperature, and Mach number along the nozzle centerline using **Plot Over Line**.
2. Repeat Q.1 but change the static pressure at the exit of the nozzle to detect the flow regime represented by curve (a) in Figure 22.1.
3. Repeat Q.1 but change the static pressure at the exit of the nozzle to detect the flow regime represented by curve (b) in Figure 22.1.
4. Repeat Q.1 but change the static pressure at the exit of the nozzle to detect the flow regime represented by curve (d) in Figure 22.1.
5. Compare and comment on the results of different flow regimes.
6. Download the oblique shock mesh and configuration files provided at the start of this tutorial. Run the case for the oblique shock problem. (*Hint: You need to stabilize the solution by tuning CFL number or finding the proper Riemann solver.*)
 - (a) Create coloured contours of Pressure, Temperature, and Mach number.
 - (b) Plot non-dimensionalized Pressure, Temperature, and Mach number along the centerline of the domain using **Plot Over Line**.
 - (c) Comment on the results.

Bibliography

Articles

- [2] Jorge E. Bardina, Peter G. Huang, and Thomas J. Coakley. “Turbulence Modeling Validation, Testing, and Development”. In: (1997) (cited on page 45).
- [3] Madeleine Coutanceau and Roger Bouard. “Experimental determination of the main features of the viscous flow in the wake of a circular cylinder in uniform translation. Part 1. Steady flow”. In: *Journal of Fluid Mechanics* 79.2 (1977), pages 231–256 (cited on page 179).
- [5] SK Godunov. “Different methods for shock waves”. In: *Moscow State University* (1954) (cited on page 90).
- [6] Ami Harten. “High resolution schemes for hyperbolic conservation laws”. In: *Journal of computational physics* 135.2 (1997), pages 260–278 (cited on page 90).
- [8] Osamu Inoue and Nozomu Hatakeyama. “Sound generation by a two-dimensional circular cylinder in a uniform flow”. In: *Journal of Fluid Mechanics* 471 (2002), pages 285–314 (cited on page 179).
- [9] W. P. Jones and Brian Edward Launder. “The Prediction of Laminarization with a Two-Equation Model of Turbulence”. In: *International journal of heat and mass transfer* 15.2 (1972), pages 301–314 (cited on page 45).
- [10] Charles L Ladson. “Effects of independent variation of Mach and Reynolds numbers on the low-speed aerodynamic characteristics of the NACA 0012 airfoil section”. In: (1988) (cited on pages 148, 204).
- [11] Charles L Ladson, Acquilla S Hill, and William G Johnson Jr. “Pressure distributions from high Reynolds number transonic tests of an NACA 0012 airfoil in the Langley 0.3-meter transonic cryogenic tunnel”. In: (1987) (cited on page 148).

- [12] Brian Edward Launder and B. I. Sharma. “Application of the Energy-Dissipation Model of Turbulence to the Calculation of Flow near a Spinning Disc”. In: *Letters in heat and mass transfer* 1.2 (1974), pages 131–137 (cited on page 45).
- [15] Stephen B Pope. “Turbulent Flows”. In: (2001) (cited on page 26).
- [16] Ludwig Prandtl. “7. Bericht Über Untersuchungen Zur Ausgebildeten Turbulenz”. In: *ZAMM- Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik* 5.2 (1925), pages 136–139 (cited on page 43).
- [17] Osborne Reynolds. “IV. On the Dynamical Theory of Incompressible Viscous Fluids and the Determination of the Criterion”. In: *Philosophical transactions of the royal society of london.(a.)* 186 (1895), pages 123–164 (cited on page 32).
- [19] V Schmitt. “Pressure distributions on the ONERA M6-wing at transonic mach numbers, experimental data base for computer program assessment”. In: *AGARD AR-138* (1979) (cited on page 193).
- [22] E. F. Toro. “Riemann Solvers with Evolved Initial Conditions”. In: *International journal for numerical methods in fluids* 52.4 (2006), pages 433–453 (cited on pages 66, 77, 79, 85, 87).
- [23] Bram Van Leer. “Towards the ultimate conservative difference scheme. II. Monotonicity and conservation combined in a second-order scheme”. In: *Journal of computational physics* 14.4 (1974), pages 361–370 (cited on page 88).
- [24] Frank M. White. “Fluid Mechanics”. In: (2015) (cited on page 13).

Books

- [1] Dale Anderson, John C. Tannehill, and Richard H. Pletcher. *Computational Fluid Mechanics and Heat Transfer*. CRC Press, 2016 (cited on pages 43, 46, 66, 95).
- [4] Peter Alan Davidson. *Turbulence: An Introduction for Scientists and Engineers*. Oxford University Press, 2015 (cited on pages 26, 31).
- [7] Charles Hirsch. *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics*. Elsevier, 2007 (cited on pages 13, 16, 66, 90, 95).
- [13] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Volume 31. Cambridge university press, 2002 (cited on pages 66, 77, 85, 88).
- [14] Bruce Roy Munson et al. *Fluid Mechanics*. Wiley Singapore, 2013 (cited on page 13).
- [21] Steven H Strogatz. *Nonlinear dynamics and chaos with student solutions manual: With applications to physics, biology, chemistry, and engineering*. CRC press, 2018 (cited on page 27).
- [25] David C Wilcox et al. *Turbulence Modeling for CFD*. Volume 2. DCW industries La Canada, CA, 1998 (cited on pages 34, 42, 46, 47).