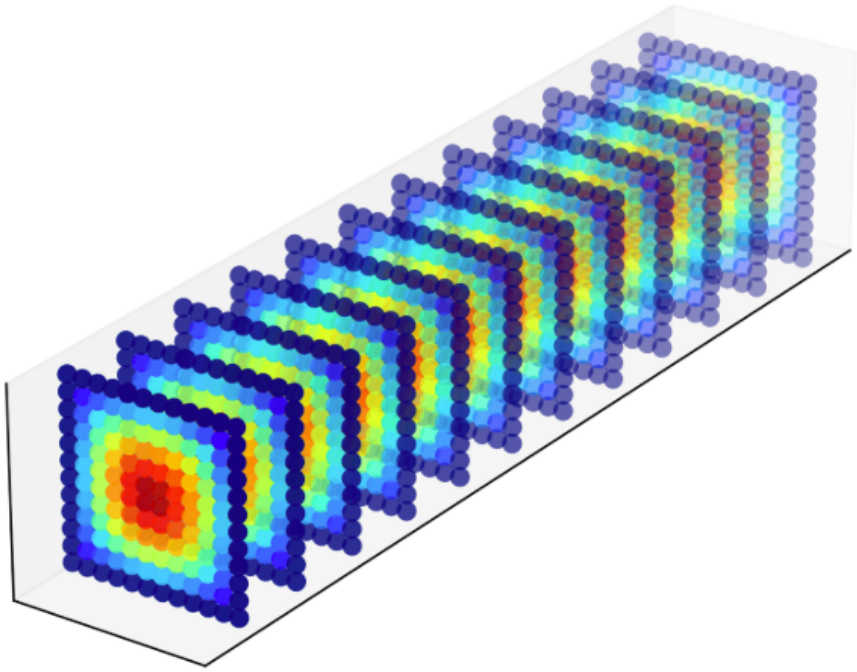
High Performance CFD Solver for Laminar Pipe Flow Problem



Panagiotis Athanasiadis

Advisor: **Vassilios Dimakopoulos**, Professor, Department of Computer Science and Engineering

University of Ioannina

School of Engineering

Ioannina 2026

Table of Contents

Glossary

Notation

Throughout this document, **vectors are written in bold**. For example, the velocity vector is written $\mathbf{u}$ rather than u, and a general vector field is written $\mathbf{F}$. Scalars are written in plain italic, e.g. pressure p or density ρ. Also matrices are denoted with bold capital letters like $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and so on. The indexing is always zero based (unless explicitly stated otherwise) and all the data structures are treated as row-major (unless explicitly stated otherwise).

CFD Computational Fluid Dynamics

COO Coordinate Format

CRS Compressed Row Storage

CSC Compressed Column Storage

CSR Compressed Sparse Row

CPU Central Processing Unit

CV Control Volume

FDM Finite Difference Method

FEM Finite Element Method

FVM Finite Volume Method

GPU Graphics Processing Unit

N–S Navier–Stokes Equations

nnz Number of non-zero entries

PDE Partial Differential Equation

Re Reynolds Number

Chapter 1

Introduction

1.1 What is CFD?

Computational Fluid Dynamics (CFD) is the numerical simulation of fluid flow by solving the governing equations of fluid dynamics on a discretized domain, combining knowledge from fluid dynamics and numerical analysis into an interdisciplinary field. Rather than relying on physical experiments or analytical solutions, which are limited to simple geometries and flow conditions, CFD provides a flexible and cost-effective approach to studying fluid behavior in complex scenarios. It has become an major tool across a wide range of engineering and scientific disciplines, including aerodynamics, biomedical engineering, environmental modeling, and energy systems, where direct measurement of flow quantities is either non feasible or prohibitively expensive.

At its core, a CFD program consists of three components: a mesh that discretizes the domain into a finite number of control volumes (CV), a discretization scheme that converts the continuous governing equations into a system of algebraic equations defined on that mesh, and a numerical solver that iteratively drives the residual of those equations to zero. The accuracy and efficiency of the simulation depend on the choices made at each of these three stages, and a significant part of CFD research is

devoted to developing and improving methods for each one.

The governing equations solved in CFD are the Navier–Stokes equations (N–S), which describe the conservation of mass and momentum in a viscous fluid. For the incompressible laminar flows considered in this study, these equations take a relatively simple form. The Finite Volume Method (FVM), used in this work, is the most popular discretization approach (among others such as Finite Elements Methods, Finite Differences Methods, Spectral Methods etc.) in CFD, thanks to its local conservation properties.

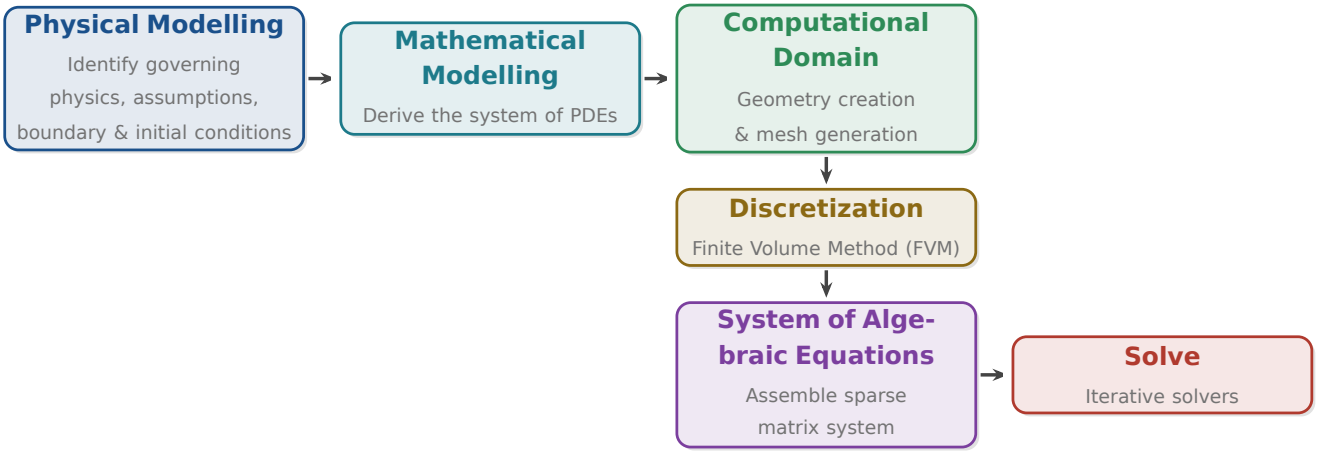


Figure 1.1: Discrete steps for numerical simulation of physical phenomena.

1.2 Laminar and Turbulent Flows

Fluid flows are broadly classified into two categories: laminar and turbulent. In **laminar flow**, fluid particles move in smooth, ordered layers with no lateral mixing between them, and the flow behavior is entirely governed by viscous forces. Laminar flow is commonly encountered in blood flow in small vessels, honey pouring smoothly from a bottle. In **turbulent flow**, the motion becomes chaotic and irregular, characterized by the presence of eddies and vortices with inertial forces dominating over viscous ones. Turbulent flow is the prevailing cetegory in most real life applications, including flow over aircraft wings, flow in pipe networks and combustion chambers. The transition between the two regimes is governed by the Reynolds number (see 3.25).

The two categories present different challenges for both analysis and numerical sim-

ulation. Laminar flows are well described by the Navier–Stokes (see 3.4.1) equations and can be solved directly without any extra modeling assumptions. Turbulent flows, on the other hand involve a vast range of interacting scales that are computationally difficult to solve at high Reynolds numbers, requiring the introduction of turbulence models that introduce additional modeling assumptions. The flows considered in this work are in the laminar category, allowing the N–S to be solved directly without any turbulence modeling.

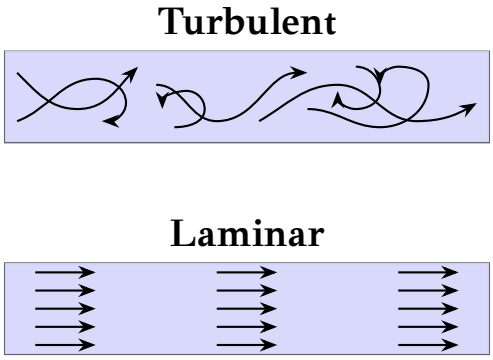


Figure 1.2: Comparison of fluid flows in a pipe. The top section illustrates Turbulent flow with complex, irregular, and swirling streamlines. The bottom section shows Laminar flow, where the fluid moves in smooth, parallel layers that are not mixing with each other.

1.3 Scope and Objectives of this Study

This report begins by establishing the physical and mathematical foundations required to understand and implement a three-dimensional incompressible laminar flow solver (simulating fluid flow through a rectangular pipe) by covering vector calculus, linear algebra, sparse matrices, and the derivation of the incompressible N–S from the conservation laws of fluid dynamics.

The discretization happens on a structured collocated mesh, and the resulting system of algebraic equations is solved using the Levenberg–Marquardt method.

A central objective of this study is the efficient implementation of the solver in C++, with all computationally intensive operations accelerated on Graphics Processing Unit (GPU) using the CUDA programming model. Specifically, the sparse system arising at each Levenberg–Marquardt iteration is solved using the NVIDIA cuDSS library,

the evaluation of the governing equations, the computation of the sparse Jacobian matrix, and all additional linear algebra operations are also performed entirely on the GPU, leveraging cuBLAS and cuSPARSE, making the entire solution pipeline GPU-accelerated.

The specific objectives of this study are summarized as follows

- Presentation of the physical and mathematical foundations of fluid dynamics, including the derivation of the incompressible N–S.
- Discretization of the three-dimensional incompressible N–S using the FVM on a structured collocated mesh.
- Assembly of the global residual function and computation of the sparse Jacobian matrix using numerical differentiation, both evaluated entirely on the GPU.
- Implementation of the Levenberg–Marquardt iterative solver with a fully GPU-accelerated solution pipeline, leveraging CUDA, cuDSS, cuBLAS and cuSPARSE.

The complete source code of this study is openly available on Github, accompanied by documentation generated, making the implementation fully reproducible and accessible to anyone.

Chapter 2

Preliminaries: Mathematical Foundations

2.1 Vector Calculus

As a fluid flows, the variables characterizing it's behavior change across space and time. Vector calculus provides the essential tools to describe these dynamics, along with the fundamental theorems which help to interpret the underlying physics.

2.1.1 Fields

Scalar Field

A **scalar field** is a function that assigns a single real number (a scalar) to every point in some region of space. Formally, a scalar field in $\mathbb{R}^n$ is a map

$$f : \mathbb{R}^n \to \mathbb{R}, \tag{2.1}$$

so that each point $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is associated with a value $f(\mathbf{x}) \in \mathbb{R}$

Vector Field

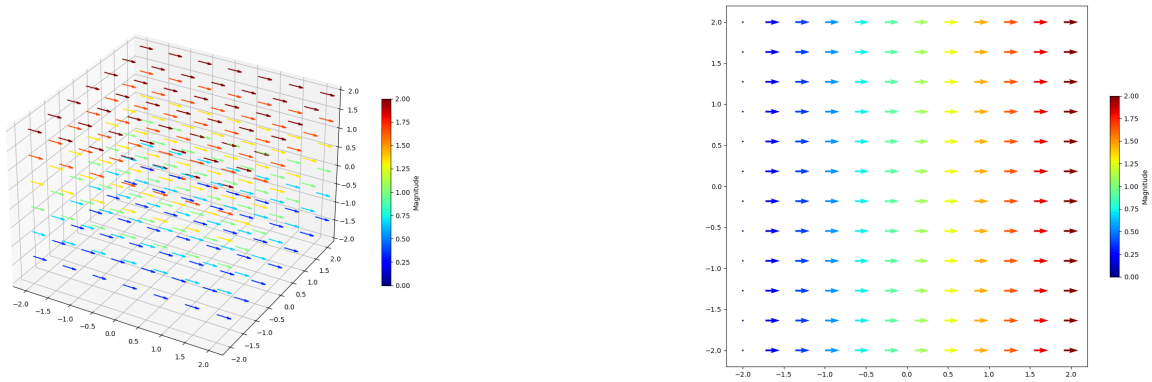
A **vector field** is a function that assigns a vector to every point in space. In $\mathbb{R}^n$, a vector field is a map

$$\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n, \tag{2.2}$$

so each point $\mathbf{x}$ is associated with a vector $\mathbf{F}(\mathbf{x})$. In three dimensions, this takes the form

$$\mathbf{F}(x, y, z) = F_x(x, y, z)\,\mathbf{i} + F_y(x, y, z)\,\mathbf{j} + F_z(x, y, z)\,\mathbf{k}, \tag{2.3}$$

where F_x, F_y, and F_z are scalar-valued component functions and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are unit vectors in the directions of the x-axis, y-axis, and z-axis respectively.



(a) A vector field in 3 dimensions.

(b) A vector field in 2 dimensions.

Figure 2.1: Example of vector fields in multiple dimensions. The multicolor scheme represents the magnitude of each vector.

2.1.2 The Nabla Operator : Divergence and Curl

The Nabla Operator

The nabla operator, also known as "del", is a vector differential operator. In three-dimensional Cartesian coordinates, it is defined as:

$$\nabla = \mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z}. \tag{2.4}$$

Divergence

The divergence of a differentiable vector field $\mathbf{F}(x, y, z) = F_x\mathbf{i} + F_y\mathbf{j} + F_z\mathbf{k}$ is the dot product of the nabla operator and the vector field.

It results in a **scalar field** that measures the magnitude of a vector field's source or sink at a given point (i.e., how much the field is expanding or contracting).
Defined as:

$$\text{div}\mathbf{F} = \nabla \cdot \mathbf{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}. \tag{2.5}$$

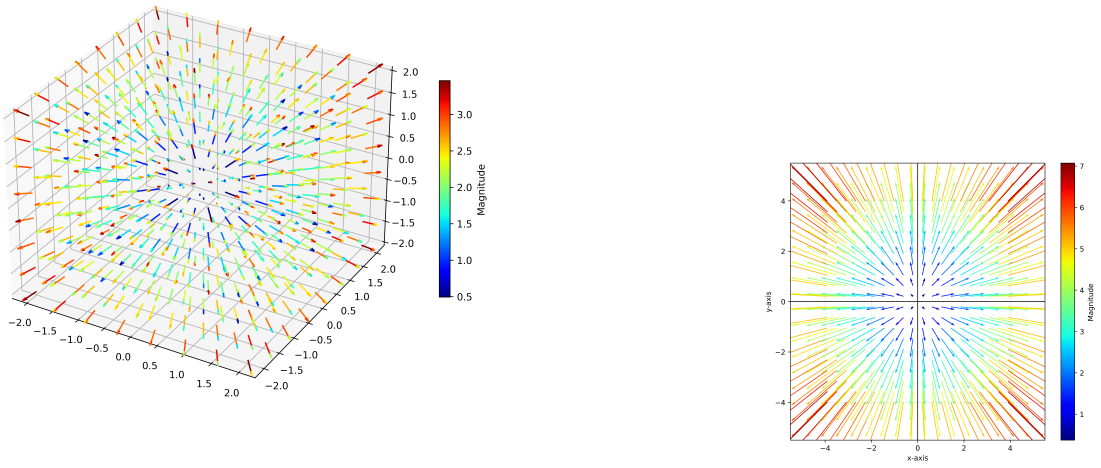


Figure 2.2: Examples of vector fields in multiple dimensions, showcasing what is called a "source" (vectors point outward from the origin, meaning $\nabla \cdot \mathbf{F} > 0$). The multicolor scheme represents the magnitude of each vector.

Curl

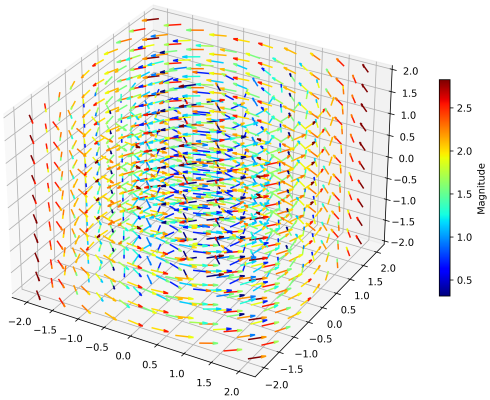
The curl of a vector field $\mathbf{F}$ is the cross product of the nabla operator and the vector field.

It results in a **vector field** that represents the infinitesimal rotation or "circulation" of the field in 3D space. The direction of the resulting vector is given by the right-hand rule. It can be computed using a formal determinant:

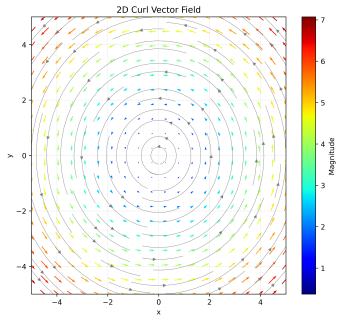
$$\text{curl}\mathbf{F} = \nabla \times \mathbf{F} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ F_x & F_y & F_z \end{vmatrix}. \tag{2.6}$$

Expanding this determinant yields:

$$\nabla \times \mathbf{F} = \left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z}\right)\mathbf{i} + \left(\frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x}\right)\mathbf{j} + \left(\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y}\right)\mathbf{k}. \tag{2.7}$$



(a) Curl of a vector field in 3 dimensions.



(b) Curl of a vector field in 2 dimensions. Grey arrows represent the rotation of the curl.

Figure 2.3: The positive curls of vector fields in different dimensions. The multicolor scheme represents the magnitude of each vector.

2.1.3 Divergence Theorem

The divergence theorem implies that the net flux of a vector field through a closed surface is equal to the total volume of all sources and sinks over the region inside the surface. Mathematically can be expressed as

$$\iiint_V (\nabla \cdot \mathbf{F})\, dV = \oiint_S \mathbf{F} \cdot \hat{\mathbf{n}}\, dS, \tag{2.8}$$

where:

- V is a volume in three-dimensional space,
- S is a closed surface on the boundary of V
- $\hat{\mathbf{n}}$ is a normal unit vector on S pointing outwards
- $\mathbf{F}$ is a vector field defined on V

The theorem is useful in the integration of the governing equations (see 5.2.2).

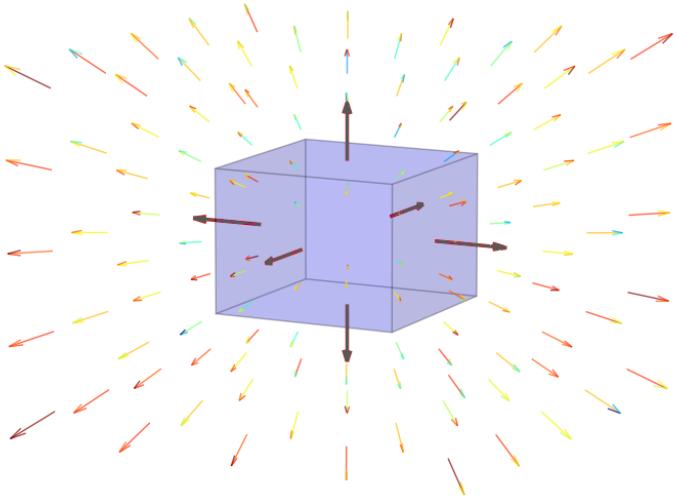


Figure 2.4: An example of closed surface S (shaded in blue) bounding a cuboid volume V is shown with its outward-pointing normal unit vectors $\hat{\mathbf{n}}$ (gray arrows). The surrounding vector field $\mathbf{F}$ spreads throughout the space, with the color of the field lines corresponding to the vector magnitude (ranging from blue for lower magnitudes to red for higher magnitudes).

2.2 Linear Algebra

2.2.1 Matrices and Matrix Operations

Matrix Addition

Two matrices $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ of the same size $m \times n$ can be added element-wise:

$$\mathbf{A} + \mathbf{B} = [a_{ij} + b_{ij}] \tag{2.9}$$

Multiplication by a Scalar

A matrix $\mathbf{A} = [a_{ij}]$ multiplied by a scalar $s \in \mathbb{R}$ scales every element:

$$s\mathbf{A} = [sa_{ij}]\,. \tag{2.10}$$

Matrix Multiplication

The product $\mathbf{C} = \mathbf{AB}$ of an $m \times n$ matrix $\mathbf{A} = [a_{ij}]$ and an $n \times p$ matrix $\mathbf{B} = [b_{ij}]$ is an $m \times p$ matrix $\mathbf{C} = [c_{ij}]$ whose entries are:

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}. \tag{2.11}$$

Transpose

The transpose of an $m \times n$ matrix $\mathbf{A} = [a_{ij}]$ is the $n \times m$ matrix $\mathbf{A}^\top$ obtained by reflecting $\mathbf{A}$ across its main diagonal:

$$\mathbf{A}^\top = [a_{ji}]\,. \tag{2.12}$$

Matrix–Vector Product

A matrix $\mathbf{A} = [a_{ij}]$ of size $m \times n$ can act on a column vector $\mathbf{x} \in \mathbb{R}^n$ to produce a new vector $\mathbf{y} \in \mathbb{R}^m$:

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \qquad y_i = \sum_{j=1}^{n} a_{ij}\, x_j. \tag{2.13}$$

2.2.2 Square Matrices

Symmetric and Antisymmetric

A square matrix $\mathbf{A} = [a_{ij}]$ is called **symmetric** if it equals its own transpose,

$$\mathbf{A} = \mathbf{A}^\top, a_{ij} = a_{ji} \quad \text{for every } i, j. \tag{2.14}$$

This means the matrix is a mirror image of itself across the main diagonal.

A square matrix $\mathbf{A} = [a_{ij}]$ is called **antisymmetric** (or **skew-symmetric**) if it equals the negative of its transpose,

$$\mathbf{A} = -\mathbf{A}^\top, a_{ij} = -a_{ji} \quad \text{for every } i, j. \tag{2.15}$$

Diagonal and Identity

A square matrix $\mathbf{D} = [d_{ij}]$ is called **diagonal** if all entries outside the main diagonal are zero:

$$d_{ij} = 0 \quad \text{for every } i \neq j. \tag{2.16}$$

The **identity** matrix $\mathbf{I}$ is a diagonal matrix with every diagonal entry equal to one. Can be defined as:

$$(\mathbf{I})_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \tag{2.17}$$

The Inverse of a matrix

A square matrix $\mathbf{A} = [a_{ij}]$ is **invertible** (non-singular) if there exists a matrix $\mathbf{A}^{-1}$ such that

$$\mathbf{A}\,\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}. \tag{2.18}$$

The inverse exists if and only if the determinant is non-zero:

$$\mathbf{A} \text{ is invertible } \iff \det(\mathbf{A}) \neq 0. \tag{2.19}$$

Upper and Lower Triangular Matrices

A square matrix $\mathbf{U} = [u_{i,j}]$ is called **upper triangular** if all entries *below* the main diagonal are zero:

$$u_{ij} = 0 \quad \text{for every } i > j. \tag{2.20}$$

A square matrix $\mathbf{L} = [l_{i,j}]$ is called **lower triangular** if all entries *above* the main diagonal are zero:

$$l_{ij} = 0 \quad \text{for every } i < j. \tag{2.21}$$

2.2.3 Describing a System of Equations using Matrices

Consider a system of n linear equations in n unknowns $x_1, x_2, \ldots, x_n$:

$$\begin{aligned} a_{11}\,x_1 + a_{12}\,x_2 + \cdots + a_{1n}\,x_n &= b_1, \\ a_{21}\,x_1 + a_{22}\,x_2 + \cdots + a_{2n}\,x_n &= b_2, \\ &\vdots \\ a_{n1}\,x_1 + a_{n2}\,x_2 + \cdots + a_{nn}\,x_n &= b_n. \end{aligned} \tag{2.22}$$

This system can be written compactly as a single matrix equation:

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}, \tag{2.23}$$

where $\mathbf{A}$ is the $n \times n$ **coefficient matrix**, $\mathbf{x}$ is the **unknown vector**, and $\mathbf{b}$ is the **right-hand side vector**:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}. \tag{2.24}$$

Each row of $\mathbf{A}$ corresponds to one equation, and each column corresponds to one unknown.

2.2.4 Euclidean norm

Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)^\top \in \mathbb{R}^n$. The **Euclidean norm** (or ℓ^2 norm) of $\mathbf{x}$ is defined as

$$\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{\mathbf{x}^\top \mathbf{x}}. \tag{2.25}$$

It measures the straight-line distance from the origin $\mathbf{0}$ to the point $\mathbf{x}$ in $\mathbb{R}^n$. The squared norm is often more convenient to work with:

$$\|\mathbf{x}\|^2 = \mathbf{x}^\top \mathbf{x} = \sum_{i=1}^{n} x_i^2. \tag{2.26}$$

2.3 Sparse Matrices

Square sparse matrices arise naturally in many applications, especially in CFD. Storing all n^2 entries would be enormously wasteful and sometimes straight up impossible, instead specialized sparse storage formats retain only the non-zero values and their locations.

2.3.1 Definition

A matrix $\mathbf{A}$ of size $n \times n$ is called **sparse** if the vast majority of its entries are zero. More precisely, if $\text{nnz}(\mathbf{A})$ denotes the number of non-zero entries in matrix $\mathbf{A}$, then $\mathbf{A}$ is sparse when

$$\text{nnz}(\mathbf{A}) \ll n^2. \tag{2.27}$$

2.3.2 Sparsity ratio

The **sparsity ratio** is the fraction between zero entries and total entries:

$$\text{sparsity} = 1 - \frac{\text{nnz}(\mathbf{A})}{n^2}. \tag{2.28}$$

2.3.3 Coordinate Format

The **Coordinate format** (COO), is the most intuitive sparse storage scheme. It stores three arrays of length $\text{nnz}(\mathbf{A})$:

$$\texttt{val}[k], \quad \texttt{row}[k], \quad \texttt{col}[k], \qquad k = 0, 1, \ldots, \text{nnz} - 1, \tag{2.29}$$

where $\texttt{val}[k]$ is the value of the k-th non-zero entry, located at row $\texttt{row}[k]$ and column $\texttt{col}[k]$ (using zero-based indexing). Entries may be stored in any order, though row-major order is conventional.

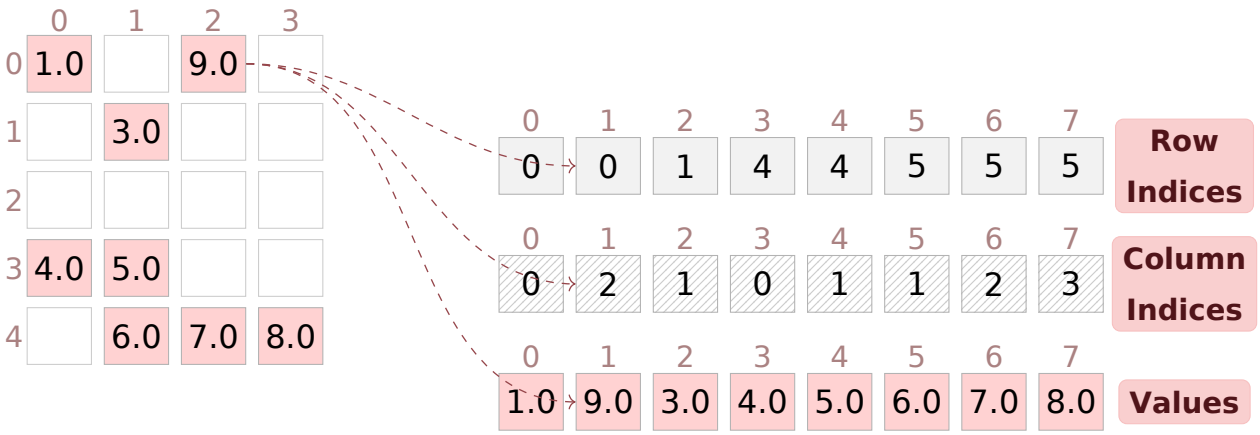


Figure 2.5: Coordinate Format (COO) representation of a 5×4 sparse matrix using zero-based indexing. Non-zero values are highlighted in red. The three arrays **Row Indices**, **Column Indices**, and **Values** each have length $\text{nnz} = 8$. Each triplet $(\texttt{row}[k], \texttt{col}[k], \texttt{val}[k])$ encodes the position and value of the k-th non-zero entry. Dashed arrows illustrate how the element 9.0 at position $(0, 2)$ maps to position 1 in all three arrays.

Given an entry in the COO format (zero-based), the corresponding position in the dense one dimensional matrix (row-major) is computed by

$$\underbrace{\texttt{row\_indices}[k]}_{\text{row position}} \times \underbrace{\texttt{leading\_dimension}}_{\text{row stride}} + \underbrace{\texttt{column\_indices}[k]}_{\text{column position}}, \tag{2.30}$$

- `leading_dimension` is the number of columns in the dense matrix.

2.3.4 Compressed Row Storage

The **Compressed Row Storage** format (CRS), also known as **CSR** (Compressed Sparse Row), eliminates the redundancy in the COO row array by replacing it with a **row pointer** array. It uses three arrays:

$$\texttt{val}[k], \quad \texttt{col\_idx}[k], \quad \texttt{row\_ptr}[l] \qquad k = 0, 1, \ldots, \text{nnz} - 1 \text{ and } l = 0, 1, \ldots, n. \tag{2.31}$$

The arrays `val` and `col_idx` store the non-zero values and their column indices, in row-major order. The key compression is in `row_ptr`: its i-th entry gives the index

in `val` where row i begins, and its last entry equals nnz.
The non-zeros of row i are therefore located at positions

$$\texttt{val}\big[\,\texttt{row\_ptr}[i],\ldots,\texttt{row\_ptr}[i{+}1]{-}1\,\big] = \begin{cases} \text{row } i \text{ is empty} & \text{if } \texttt{row\_ptr}[i] = \texttt{row\_ptr}[i+1] \\ \text{entries of row } i & \text{otherwise} \end{cases} \tag{2.32}$$

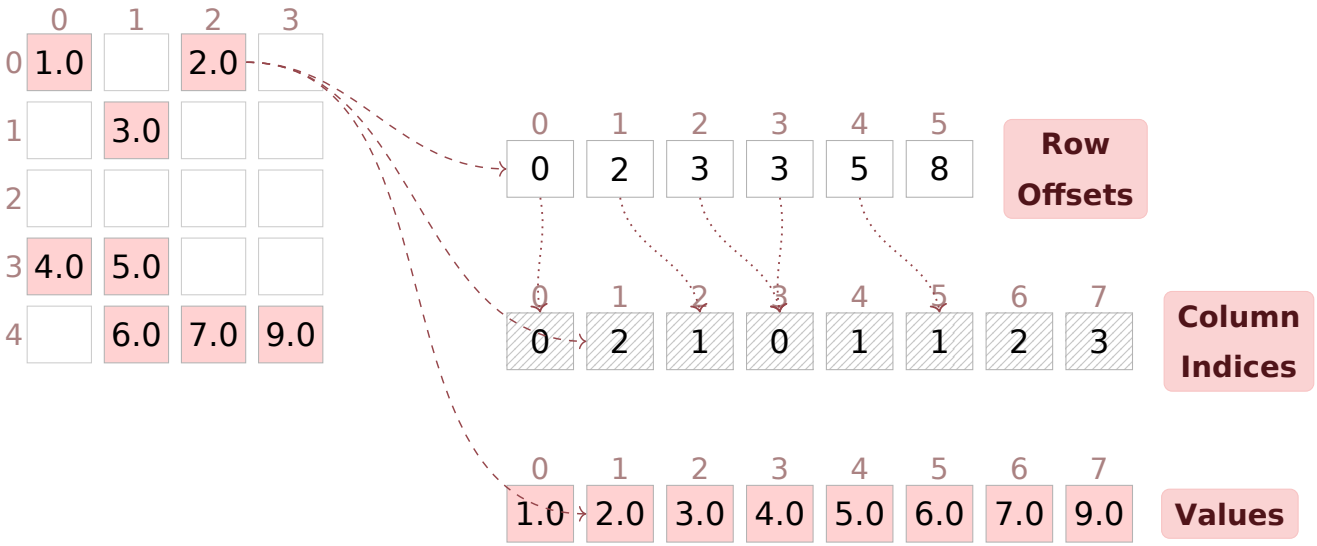


Figure 2.6: CSR format representation of a 5×4 sparse matrix using zero-based indexing. Non-zero values are highlighted in red. The **Row Offsets** array (length $n+1=6$) stores the starting position of each row in the **Column Indices** and **Values** arrays. Dotted arrows indicate how each row offset points to the corresponding start of that row's entries. Dashed arrows illustrate how the element 2.0 (at row 0, column 2) maps to position 1 in the Column Indices and Values arrays, and to position 0 in the Row Offsets array.

CRS requires $(2 \times \text{nnz} + n + 1)$ storage.

Given an entry in the CSR format (zero-based), the corresponding position in the dense matrix (row-major) is computed by

$$\underbrace{\texttt{i}}_{\text{row position}} \times \underbrace{\texttt{leading\_dimension}}_{\text{row stride}} + \underbrace{\texttt{column\_indices}\big[\,\texttt{row\_offsets}[\texttt{i}] + k\,\big]}_{\text{column position}}, \tag{2.33}$$

- `i` is the row index,

- $k \in \{0, 1, \ldots, \texttt{row\_offsets}[\texttt{i}+1] - \texttt{row\_offsets}[\texttt{i}] - 1\}$ is the local offset within `row`,
- `leading_dimension` is the number of columns in the dense matrix.

2.3.5 Compressed Column Storage

The **Compressed Column Storage** format (CSC) is the column-wise analogue of CRS. It uses three arrays:

$$\texttt{val}[k], \quad \texttt{row\_idx}[k], \quad \texttt{col\_ptr}[l] \qquad k = 0, 1, \ldots, \mathrm{nnz} - 1 \textit{ and } l = 0, 1, \ldots, n. \tag{2.34}$$

Here `val` and `row_idx` store the non-zero values and their row indices in **column-major** order. The array `col_ptr` plays the role of `row_ptr` in CRS, but for columns: its j-th entry gives the index in `val` where column j begins, and its last entry equals nnz. The non-zeros belonging to column j are located at positions:

$$\texttt{val}\big[\,\texttt{col\_ptr}[j], \ldots, \texttt{col\_ptr}[j{+}1]{-}1\,\big] = \begin{cases} \text{column } j \text{ is empty} & \text{if } \texttt{col\_ptr}[j] = \texttt{col\_ptr}[j+1] \\ \text{entries of column } j & \text{otherwise} \end{cases} \tag{2.35}$$

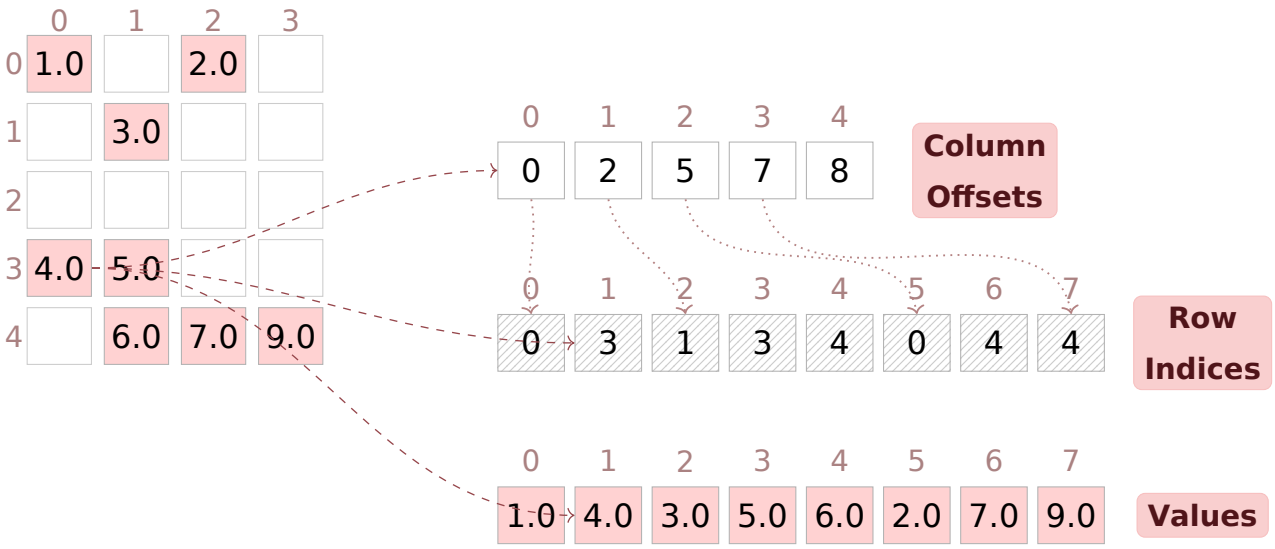


Figure 2.7: CSC format representation of a 5×4 sparse matrix using zero-based indexing. Non-zero values are highlighted in red. The **Column Offsets** array (length $n+1=5$) stores the starting position of each column in the **Row Indices** and **Values** arrays. Dotted arrows show how each column offset indexes into the start of that column's entries. Dashed arrows illustrate how the element 4.0 at position $(3, 0)$ maps to position 0 in the Column Offsets array and to position 1 in the Row indices and Values arrays.

Given an entry in the CSC format (zero-based), the corresponding position in the dense one dimensional matrix (row-major) is computed by

$$\underbrace{\texttt{row\_indices}[\texttt{column\_offsets}[j]+k]}_{\text{row position}} \times \underbrace{\texttt{leading\_dimension}}_{\text{row stride}} + \underbrace{j}_{\text{column position}}, \tag{2.36}$$

where:

- j is the column index,
- $k \in \{0, 1, \ldots, \texttt{column\_offsets}[j+1] - \texttt{column\_offsets}[j] - 1\}$ is the local offset within column j,
- `leading_dimension` is the number of columns in the dense matrix.

Note that CRS and CSC are transposes of each other in the sense that storing $\mathbf{A}$ in CSC is equivalent to storing $\mathbf{A}^\top$ in CRS:

$$\mathrm{CSC}(\mathbf{A}) \equiv \mathrm{CRS}(\mathbf{A}^\top). \tag{2.37}$$

2.4 Jacobian Matrix

2.4.1 Definition

Let $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ be a vector-valued function with components $f_1, f_2, \ldots, f_m$, each depending on the variables $x_1, x_2, \ldots, x_n$:

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(x_1, \ldots, x_n) \\ f_2(x_1, \ldots, x_n) \\ \vdots \\ f_m(x_1, \ldots, x_n) \end{pmatrix}. \tag{2.38}$$

The **Jacobian matrix** of $\mathbf{f}$ at a point $\mathbf{x}$ is the $m \times n$ matrix of all first-order partial derivatives:

$$\mathbf{J_f}(\mathbf{x}) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} & \cdots & \dfrac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \dfrac{\partial f_m}{\partial x_2} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{pmatrix}. \tag{2.39}$$

In index notation, the (i, j) entry of the Jacobian is

$$(\mathbf{J_f})_{ij} = \frac{\partial f_i}{\partial x_j}, \qquad i = 1, \ldots, m, \quad j = 1, \ldots, n. \tag{2.40}$$

Chapter 3

The Physics of Fluid Dynamics

There are many physical laws that govern the behavior of fluids but this chapter will be limited to those concerning incompressible laminar flow of a Newtonian fluid. Specifically:

- **Eulerian description**: Flow quantities are described at fixed points in space rather than by following individual fluid elements.
- **Steady-state flow**: Meaning that all flow quantities are independent of time.
- **Newtonian fluid**: The dynamic viscosity (μ) of the fluid is constant and it is independent of the shear rate.
- **Incompressible flow**: The density (ρ) of the fluid is constant.
- **Laminar flow**: The flow is smooth and ordered with no turbulent fluctuations.
- **Three-dimensional**: All quantities are defined over the three spatial dimensions using the Cartesian coordinate system.
- **No external forces**: All external body forces such as gravity are omitted.

- **Dimensionless**: All quantities are free from units of measure.

Each restriction is introduced thoroughly in the following sections.

3.1 Flow kinematics

3.1.1 Eulerian and Lagrangian descriptions

Two fundamental perspectives exist for observing a fluid in motion:

- In the **Lagrangian** description the observer travels with the fluid, following the trajectory of each individual fluid particle as it moves through space.
- In the **Eulerian** description the observer stands at a fixed point in space and records whatever fluid particles passes through it.

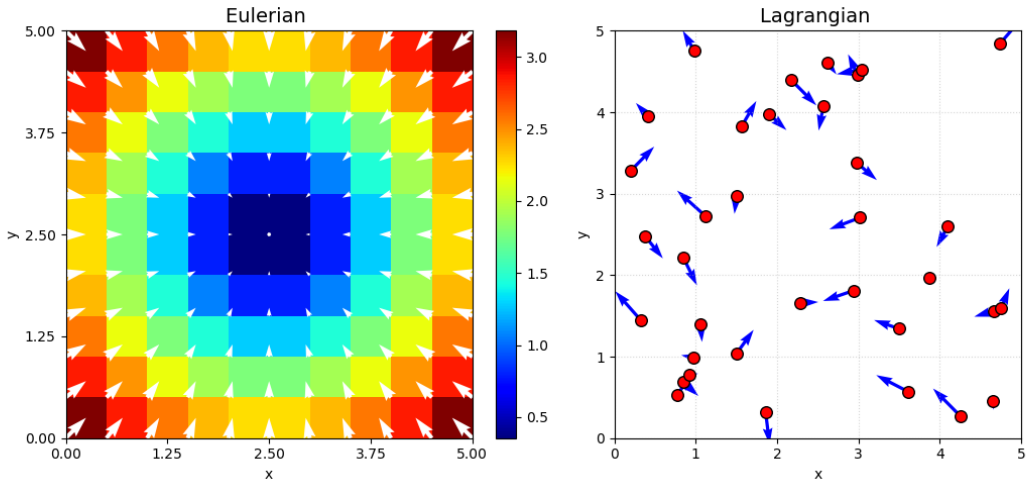


Figure 3.1: A visual example of an Eulerian description on the left on a grid, each square on the grid represents the space that is being observed by the observer, while the white arrows represent the velocity of the fluid passing through each square, the coloring are for illustration purposes. The visual example of an Lagrangian description on the right is tracking the position and velocity (blue arrows) of each individual fluid particle (red dots).

3.1.2 The Velocity Field

A fluid is a continuous medium that deforms continuously under any applied shear stress. In order to describe its motion, fluid dynamics assigns a **velocity vector** to every point in the flow domain $\Omega \subset \mathbb{R}^3$. This assignment defines the **velocity field**,

$$\mathbf{u}(\mathbf{x}) \;=\; u(\mathbf{x})\,\mathbf{i} \;+\; v(\mathbf{x})\,\mathbf{j} \;+\; w(\mathbf{x})\,\mathbf{k}, \tag{3.1}$$

where u, v, and w are the scalar velocity components along the Cartesian axes x, y, and z, and $\mathbf{x} = (x,\, y,\, z) \in \Omega$ denotes a fixed spatial position.

The magnitude of the velocity field at any point is the **flow speed**,

$$|\mathbf{u}| \;=\; \sqrt{u^2 + v^2 + w^2}\,, \tag{3.2}$$

which is a scalar field.

3.2 Conservation Laws

The behavior of fluid is governed by three fundamental laws of classical physics: the conservation of mass, the conservation of momentum, and the conservation of energy (omitted since the simulation only tracks the pressure and velocity of fluid, not the temperature). They are expressed in their differential forms, which are strictly local (they deal with quantities at a point in space). This locality makes them more intuitive to program them.

A useful concept in fluid dynamics is the fixed **control volume** (CV). It represents an arbitrary fixed region of space in which fluid is allowed to flow freely and the conservation laws can be applied there without the need to apply them to individual fluid particles.

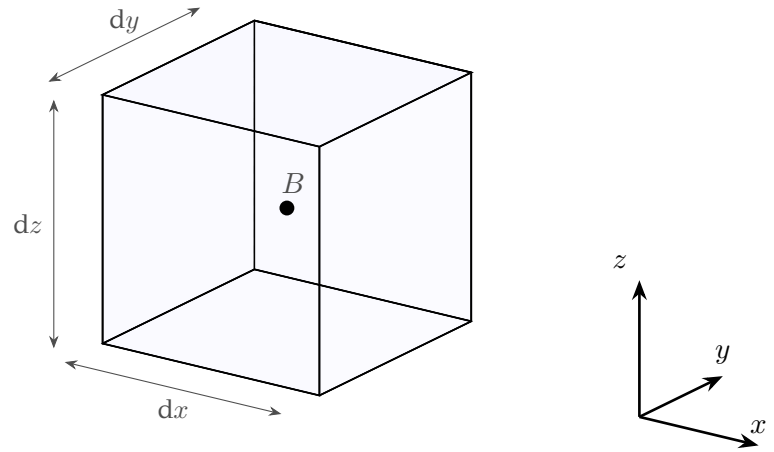


Figure 3.2: Cuboid CV with length dx,dy,dz. B represents an arbitrary point inside the control volume.

3.2.1 Conservation of Mass (Continuity Equation)

The conservation of mass states that mass can neither be created nor destroyed. To derive the continuity equation in its differential form, consider a cuboid CV like the one in Figure 3.2 which has volume

$$dV = dx \cdot \ dy \cdot dz, \tag{3.3}$$

and mass

$$dm = \rho \cdot dV, \tag{3.4}$$

where ρ is a constant that represents the **fluid density** (since it is a constant it can be eliminated when differentiating in the following equations). The net mass flow rate through the CV is the sum of the mass rates through each pair of opposite faces of the rectangle.

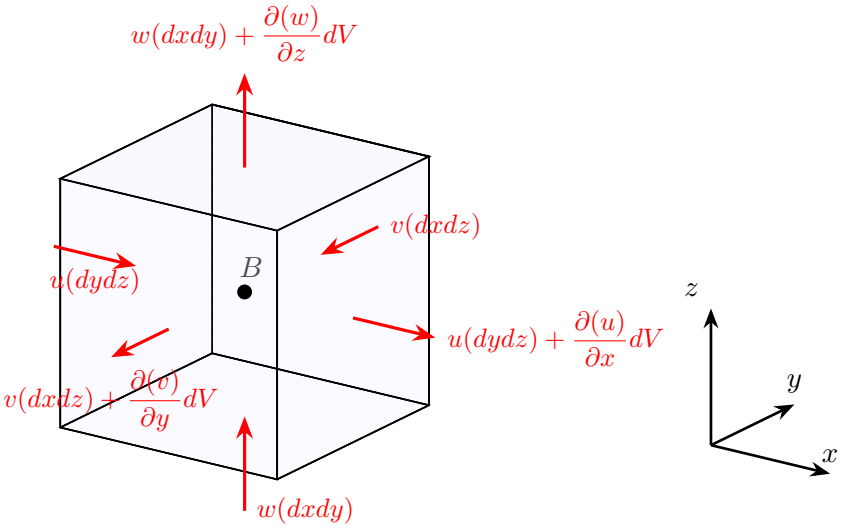


Figure 3.3: Cuboid CV showing the mass flux (red arrows) entering and leaving through each pair of opposite faces.

Lets consider the mass flow along the y direction through the front face of the CV with area $dA_y = dxdz$,

$$\textit{Mass flow rate into } dV \textit{ in the } y \textit{ direction} = vdA_y, \tag{3.5}$$

recall that v is the velocity component of the flow in the y direction (see (3.1). Similarly for the back face,

$$\textit{Mass flow rate out of } dV \textit{ in the } y \textit{ direction} = vdA_y + \frac{\partial(v)}{\partial y}dV. \tag{3.6}$$

The net mass flow rate of CV in the y axis can be found by

$$vdA_y + \frac{\partial(v)}{\partial y}dV - vdA_y = \frac{\partial(v)}{\partial y}dV. \tag{3.7}$$

With the same way, x and z axis can be found

$$\textit{Net mass flow rate of } dV \textit{ in the } x \textit{ direction} = \frac{\partial(u)}{\partial x}dV, \tag{3.8}$$

$$\textit{Net mass flow rate of } dV \textit{ in the } z \textit{ direction} = \frac{\partial(w)}{\partial z}dV. \tag{3.9}$$

Now, consider that the time rate of change of mass inside the CV is,

$$\frac{\partial \rho}{\partial t}dV, \tag{3.10}$$

and since time is omitted $\frac{\partial \rho}{\partial t} = 0$.

If mass is conserved, then the net mass flow rate out of the CV is equal to the time rate of decrease of mass inside the CV,

$$\left(\frac{\partial(u)}{\partial x}+\frac{\partial(v)}{\partial y}+\frac{\partial(w)}{\partial z}\right)dV=-\frac{\partial\rho}{\partial t}dV \tag{3.11}$$

Simplifying gives,

$$\left(\frac{\partial(u)}{\partial x}+\frac{\partial(v)}{\partial y}+\frac{\partial(w)}{\partial z}\right)=0, \tag{3.12}$$

which must be equal to zero, since mass is conserved. In the differential form the continuity equation using vector notation can be written as,

$$\boxed{\nabla\cdot(\mathbf{u})=0}. \tag{3.13}$$

3.2.2 Conservation of Momentum (Euler Equations)

The conservation of momentum states that the total momentum in a closed system (free from exterior forces) remains constant. Specifically this is expressed through Newton's second law.

From classical mechanics, Newton's second law states that the net force acting on a body equals its mass times acceleration. Described in elementary physics can be written as

$$\mathbf{F}=m\cdot\mathbf{a}, \tag{3.14}$$

where **a** denotes the linear acceleration and m the mass. Applying this law (adjusted for fluids) to a CV like 3.2 can yield the momentum equations in differential form, which are also called the *Euler equations*. The Euler (momentum) equations can be expressed as the sum of pressure forces and body forces (they are not taken into consideration but for more see A.1.1) that equal the mass times acceleration:

$$\mathbf{F_p}+\mathbf{F_b}=dm\cdot\mathbf{a}. \tag{3.15}$$

The equations does take into consideration the viscosity of the fluid. The fluid viscosity term is introduced in Section 3.4.1, where the Euler equations are extended to the N–S.

Pressure Forces

The **pressure** of the fluid at a specific point in space can be expressed by a scalar function $p=p(x,y,z)$. Starting by finding the pressure forces along the x direction

and taking into consideration that the change of pressure in that direction can be expressed as $\frac{\partial p}{\partial x}$, the net pressure for the x direction can be found.
From the left side of the x direction the pressure force is,

$$p \cdot dA_x. \tag{3.16}$$

On the right side of the x direction the pressure force is,

$$p \cdot dA_x + \left(\frac{\partial p}{\partial x}\right) dV. \tag{3.17}$$

The net pressure force for the x direction is,

$$dF_x = pdA_x - \left(pdA_x + \left(\frac{\partial p}{\partial x}\right)\right) dV = -\left(\frac{\partial p}{\partial x}\right) dV. \tag{3.18}$$

Similarly for the y and z directions,

$$dF_y = -\left(\frac{\partial p}{\partial y}\right) dV \tag{3.19}$$

$$dF_y = -\left(\frac{\partial p}{\partial z}\right) dV. \tag{3.20}$$

Finally, the net pressure force for the CV can be expressed as

$$dF = -\left(\frac{\partial p}{\partial x}\mathbf{i} + \frac{\partial p}{\partial y}\mathbf{j} + \frac{\partial p}{\partial z}\mathbf{k}\right) dV, \tag{3.21}$$

or using the vector notation,

$$\boxed{-\nabla p dV}. \tag{3.22}$$

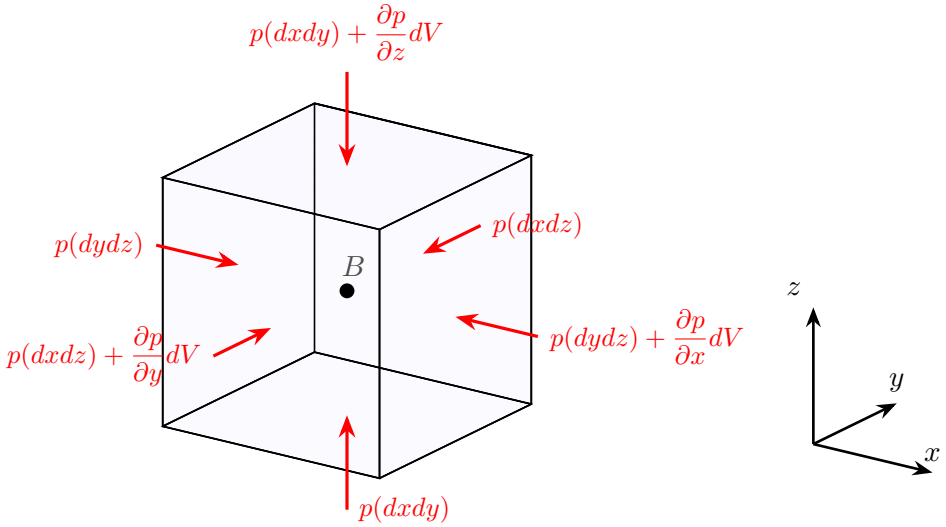


Figure 3.4: Cuboid CV showing the pressure forces (red arrows) for each pair of opposite faces.

Momentum Terms

The momentum terms for the x, y, z directions are:

$$\rho\left(u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}\right)=-\frac{\partial p}{\partial x} \qquad (x\text{-direction})$$

$$\rho\left(u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}\right)=-\frac{\partial p}{\partial y} \qquad (y\text{-direction})$$

$$\rho\left(u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}\right)=-\frac{\partial p}{\partial z} \qquad (z\text{-direction})$$

The Euler equation in vector form:

$$\boxed{\rho(\mathbf{u}\cdot\nabla)\mathbf{u}=-\nabla p}, \tag{3.23}$$

where $(\mathbf{u}\cdot\nabla)\mathbf{u}$ is the **convective acceleration** of the fluid. It describes how the velocity of a fluid particle changes as it moves through space and $-\nabla p$ is the **pressure gradient** that describes that the fluid accelerates from regions of high pressure toward regions of low pressure.

3.3 Fluid Properties

In the study of fluid mechanics, understanding how a fluid behaves under various conditions is fundamentally linked to its inherent properties. The specific properties of a fluid dictate how it will move and interact with its environment.

3.3.1 Viscosity

Viscosity, which governs a fluid's internal resistance to flow plays a central role in distinguishing how different fluids behave under stress. In order to understand viscosity better, the concept of shear stress needs to be introduced. An intuitive way to understand shear stress in fluids, is to think that a fluid that flows (in a pipe for example) can be partitioned (in parallel with the flow direction) in layers that flow with different velocities. For two layers that move relative to one another, shear stress (also known as "skin drag" and denoted with the letter τ) is developed between them because they flow with different velocities (see Figure 3.5).

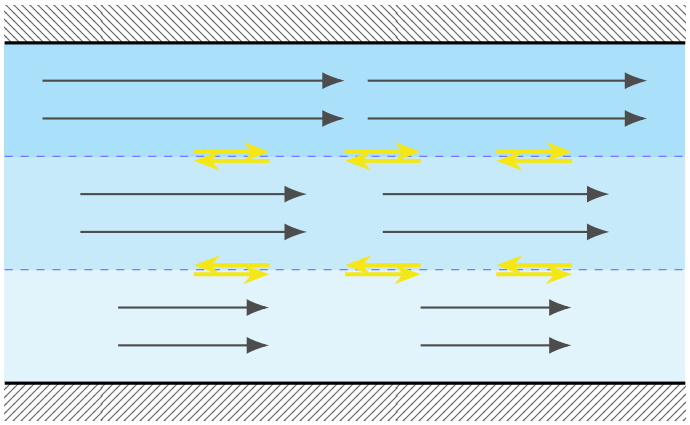


Figure 3.5: Fluid flowing in a pipe, partitioned in layers with different velocities (black arrows). The yellow arrows represents the shear stress between the layers.

One more concept that needs to be introduced is the velocity profile, which shows the distribution of fluid velocities of a flow in space. For a velocity profile similar to the simulation, a parabolic one is used.

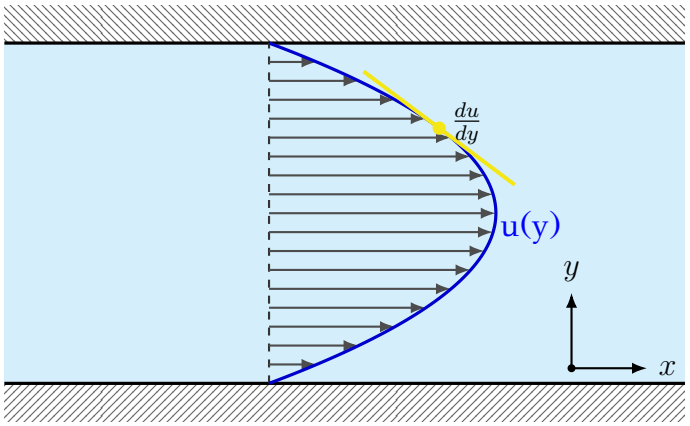


Figure 3.6: Parabolic velocity profile ($u(y)$) of laminar flow in a pipe (smooth walls) for a Newtonian fluid. The yellow point represents the velocity gradient ($\frac{du}{dy}$).

The relation between the shear stress and the velocity gradient defines the **dynamic viscosity** of a fluid and is denoted with the letter μ.

The No-Slip Condition

Looking at the figure 3.6, the velocity gradient approaches zero near the walls of the pipe. This happens because the shear stress is so massive that the velocity of the fluid

at that region is negligible, effectively **zero**. The massive shear stress is developed due to the large difference in velocity between the fluid and the wall.

Newtonian Fluids

In fluids where the relation between the shear stress and the velocity gradient is **linear** are called Newtonian fluids. This relationship is expressed through Newton's law of viscosity which is defined as

$$\boxed{\tau = \mu \frac{du}{dy}}. \tag{3.24}$$

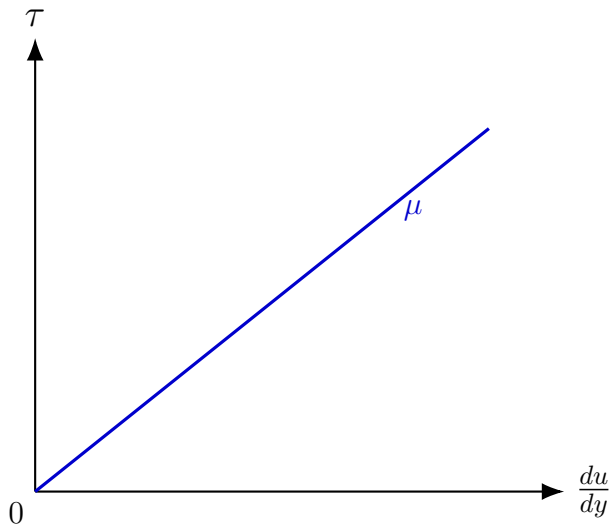


Figure 3.7: The linear relationship between shear stress and the velocity gradient for a Newtonian fluid.

3.3.2 Incompressible Fluids

Compressibility, describes how a fluid's density responds to changes in pressure. For incompressible fluids, the density (denoted with the letter ρ) of the fluid remains **constant** throughout the flow field, regardless of variations in pressure.

3.3.3 The Reynolds Number

The Reynolds number describes the relative importance of the inertial forces (the forces needed to accelerate or stop fluid motion) and the viscous forces. The Reynolds number is dimensionless, meaning it is free from units of measurement and defined as

$$\boxed{\text{Re} = \frac{\rho u_0 L}{\mu}} = \frac{\text{inertial forces}}{\text{viscous forces}}, \tag{3.25}$$

where u_0 is the characteristic velocity and L is the characteristic length which depends on the geometry in which the flow is defined. For example the characteristic length of a flow in a pipe is the internal diameter of the pipe.

The usefulness of the Reynolds number comes from the ability to describe when the flow transitions from laminar to turbulent. Specifically when the viscous forces dominate, the flow stays laminar but when the inertial forces dominate the flow transition to turbulence.

3.4 The Governing Equations

The motion of an incompressible, viscous, Newtonian fluid is governed by the incompressible *Navier–Stokes equations* (N–S), comprising the continuity equation and the momentum equation.

3.4.1 The Navier-Stokes Equations

These two equations together describe the motion of a viscous, incompressible fluid and defined as

$$\nabla \cdot \mathbf{u} = 0, \tag{3.26}$$

$$\rho(\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \mu \nabla^2 \mathbf{u}. \tag{3.27}$$

The first equation 3.26 is the **continuity equation**, it has been explained in detail in 3.2.1.

The second equation 3.27 is the **momentum equation** and it has been introduced partially by the Euler equation 3.23. Specifically unlike the Euler equation in which viscous forces are ignored, here they are expressed by the diffusion term $\mu \nabla^2 \mathbf{u}$ where $\nabla^2 \mathbf{u}$ is called the Laplacian of the velocity field. The diffusion term reflects how momentum spreads through the fluid due to viscosity. In more detail the Laplacian of the velocity field shows that momentum diffuses from regions of higher velocity to regions of lower velocity and the viscosity constant controls how quickly this diffusion happens.

Conservative and Non-Conservative Form of the Navier-Stokes Equations

The form of the N–S presented in 3.27 is called **non-conservative**. There is another form of N–S called **conservative** which is mathematically equivalent to the non-conservative form (for incompressible fluids). In conservative form, every term can be written as a divergence which will be useful in the discretization process later on (see 5.2.2).

Starting the conversion (from non-conservative to conservative) from the left hand side of momentum equation of the N–S,

$$\rho(\mathbf{u}\cdot\nabla)\mathbf{u} = \rho\nabla\cdot(\mathbf{u}\mathbf{u}), \tag{3.28}$$

which is derived using the following vector identity (ignoring ρ for simplicity)

$$\nabla\cdot(\mathbf{u}\mathbf{u}) = (\mathbf{u}\cdot\nabla)\mathbf{u} + \mathbf{u}\underbrace{(\nabla\cdot\mathbf{u})}_{=0} \quad\Longrightarrow\quad \nabla\cdot(\mathbf{u}\mathbf{u}) = (\mathbf{u}\cdot\nabla)\mathbf{u}, \tag{3.29}$$

since the fluid is incompressible ($\nabla\cdot\mathbf{u} = 0$).
Substituting back to the momentum equation

$$\rho\,\nabla\cdot(\mathbf{u}\,\mathbf{u}) = -\nabla p + \mu\nabla^2\mathbf{u} \tag{3.30}$$

and dividing by ρ on both sides

$$\nabla\cdot(\mathbf{u}\mathbf{u}) = -\frac{1}{\rho}\nabla p + \underbrace{\frac{\mu}{\rho}}_{\nu}\nabla^2\mathbf{u} \tag{3.31}$$

where ν is the kinematic viscosity.
The **conservative form** of the N-S equations are

$$\nabla\cdot\mathbf{u} = 0, \tag{3.32}$$

$$\nabla\cdot(\mathbf{u}\mathbf{u}) = -\frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u}. \tag{3.33}$$

Dimensionless Form of the Navier-Stokes Equations

As mentioned previously, in this simulation all the quantities are free from units of measure. Following the same logic, the **dimensionless form** of the N–S equations are presented

$$\nabla\cdot\mathbf{u} = 0, \tag{3.34}$$

$$\nabla\cdot(\mathbf{u}\,\mathbf{u}) = -\nabla p + \frac{1}{Re}\nabla^2\mathbf{u}. \tag{3.35}$$

3.4.2 The Million Dollar Question: Navier-Stokes Existence and Smoothness

This subsection is devoted to show the unique nature of the N–S equations. Although numerical solutions to the N–S equations have been found for a diverse range of problems regarding fluids, the theoretical understanding of those equations is still limited. Some solutions to specific cases of N–S equations have been found but still they are far from ideal. Still to this day of writing this text, the Clay Mathematics Institute offers one million American dollars to the first one that can prove that there is a global, smooth solution (or there is not) for the N–S equations in three-dimensional space and time. One of the many difficulties arising when studying the N–S equations is that they contain a nonlinear term (the convection term, see 3.23). This nonlinearity comes from the fact that the fluid's velocity influences its own rate of change and makes the equations very difficult to analyze since the mathematical tools for handling nonlinearity are still in early stage of development.

Chapter 4

Problem Definition

4.1 Fluid Flowing Through a Pipe

Fluid flow through pipes is encountered frequently in modern everyday life. At the largest scale, long-distance pipelines carry oil, natural gas, and water across entire continents. In buildings, pipe networks distribute hot and cold water, heating fluids, and refrigerants, all of which must be carefully sized to guarantee adequate flow at every outlet. Even in the human body, blood flowing through arteries and veins follows the same governing equations as a pipe flow and an understanding of these principles has guided the development of stents and artificial heart valves. In all these cases, the ability to simulate pipe flow numerically provides scientists with a powerful tool to optimize designs, reduce energy consumption, and avoid costly failures before any physical prototype is built.

4.2 Description of the Simulation

The simulation shows the spatial evolution of the velocity and pressure fields of a Newtonian, incompressible fluid flowing through a straight rectangular pipe. The flow is assumed to be steady ($\frac{\partial}{\partial t} = 0$), laminar (low RE), and governed by the

incompressible N–S equations. A specific velocity profile is imposed at the inlet, and a zero-gradient outflow condition is applied at the outlet, at the walls of the pipe the No-slip condition (see 3.3.1) is enforced.

4.2.1 Geometry

The geometry of the rectangular pipe is simple and can be described only by three parameters: length L, width W and height H. The simplicity of the geometry allows for more efficient and easy ways to discretize the space (see 5.2.1) and enforce the boundary conditions. The thickness of the walls is not considered in this simulation.

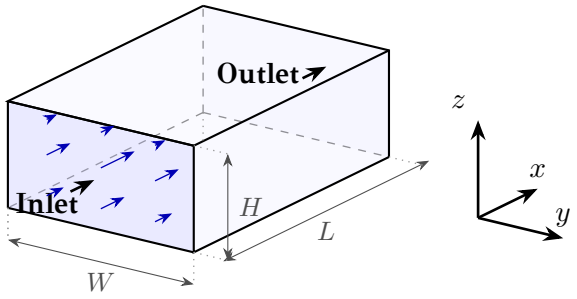


Figure 4.1: Geometry of the rectangular pipe of length L, width W and height H. The inlet face lies at $x = 0$ and the outlet at $x = L$. Blue arrows represent the direction of the flow.

4.2.2 Initial Values

At the inlet of the pipe, a parabolic velocity profile is used and it is defined as

$$u_{inlet}(y, z) = 16u_0 \left(\frac{y}{W}\right)\left(1 - \frac{y}{W}\right)\left(\frac{z}{H}\right)\left(1 - \frac{z}{H}\right), \tag{4.1}$$

where u_0 is the initial velocity.

This is a bi-parabolic where a parabola is defined both at y and z directions, specifically the factors $\left(1 - \frac{y}{W}\right)$ and $\left(1 - \frac{z}{H}\right)$ are one dimensional parabolas. When these parabolas are calculated at the wall of the pipe $(y = 0, y = W, z = 0, z = H)$ they result to zero and thus enforcing the boundary conditions. The factor of 16 ensures that the peak velocity equals to u_0 at the center of the inlet $(y = W/2, z = H/2)$.

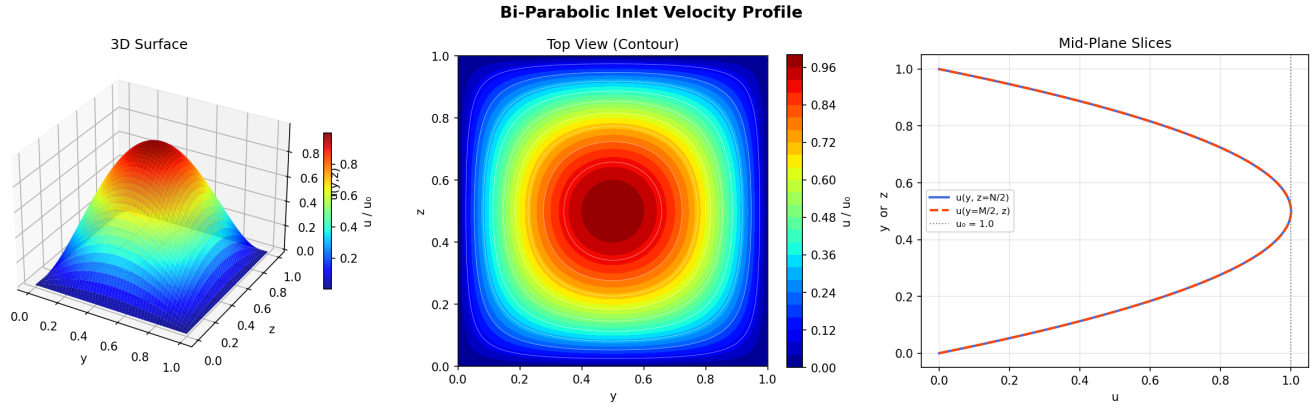


Figure 4.2: Bi-parabolic inlet velocity profile where $u_0 = 1, \quad W = 1, \quad H = 1$.
Left: A three-dimensional surface of the velocity field over the inlet cross-section. *Center:* top-view contour plot showing the velocity magnitude; the maximum $u = u_0$ is attained at the inlet center $(y{=}W/2,\, z{=}H/2)$.
Right: mid-plane profiles $u(y,\, z{=}H/2)$ and $u(y{=}W/2,\, z)$, confirming the parabolic distribution and the no-slip condition at all four walls.

Chapter 5

Discretization and The Finite Volume Method (FVM)

5.1 Why the Need for Discretization?

Discretization is a technique that transforms a continuous equation to a discrete one. The discrete form of a equation is suitable to be programmed into a computer since computers can only operate on discrete and finite sets of numbers. In the case of fluid dynamics, the physical modeling produces continuous equations (N–S equations) that they need to be discretized in ordered to be solved numerically (for most of the practical applications it is impossible or very hard to find analytical solutions). By splitting the computational domain into discrete chunks or CVs through a process called meshing and applying methods like Finite Volume Method (FVM) to each CV, the continuous problem becomes a system of algebraic equations that can be solved numerically.

5.2 The Finite Volume method

The **finite volume method** (FVM) is a popular discretization method among CFD codes, which integrates the governing equations over each non-overlapping CV, created by the meshing of the physical domain. The name "finite volume" refers to the volume of each CV.

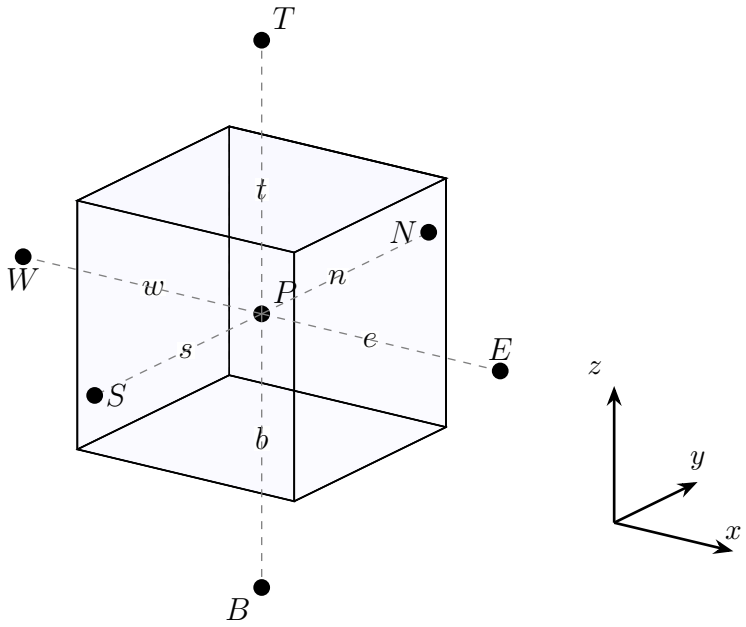


Figure 5.1: Cuboid CV Ω_P with central node P, its six face-neighbouring nodes (North N, South S, East E, West W, Top T, Bottom B), and labeled faces (North n, South s, East e, West w, Top t, Bottom b).

This process creates volume integrals for each CV that can be converted to surface integrals (using the divergence theorem) that in turn can be converted to algebraic equations and form a system of equations.

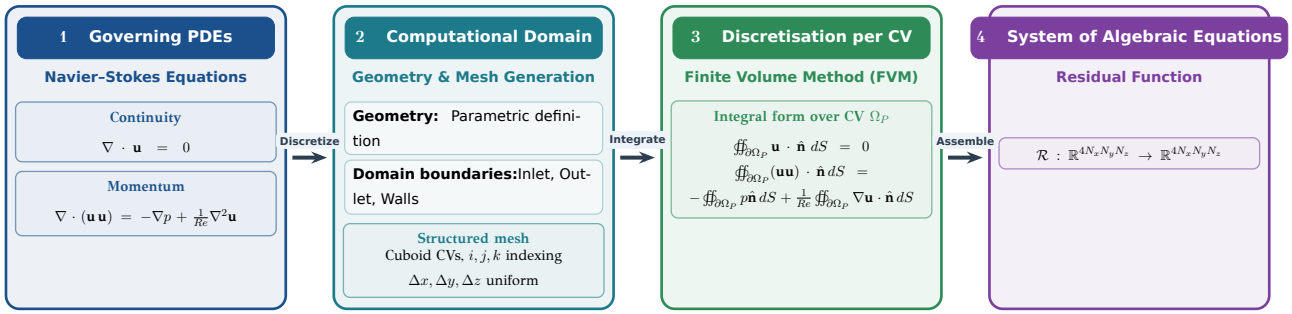


Figure 5.2: Discretization process using FVM

Specifically, fluxes that represent the flow of mass and momentum are evaluated at the faces of each cell and balanced against neighboring CVs.

The flux of ϕ through face f is defined as

$$\mathcal{F}_f = \phi_f A_f, \tag{5.1}$$

where

- ϕ_f is the transported quantity (mass, momentum) evaluated at the face.
- A_f is the corresponding face area.

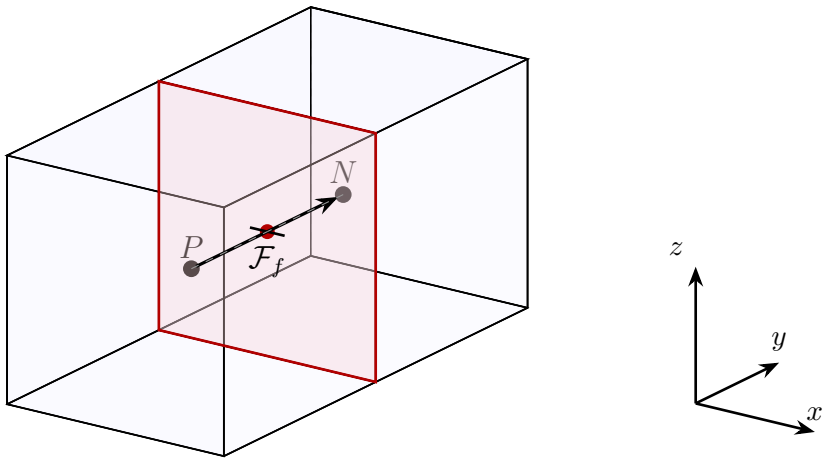


Figure 5.3: Two neighboring CVs Ω_P and Ω_N sharing a common face f (highlighted in red). The flux $\mathcal{F}_f$ is transported from node P to node W across the shared face.

To achieve this mathematically, the FVM approximates the exact surface integral of any flux over the CV surface ($\partial\Omega_P$), as a discrete sum over its faces using the **Midpoint Rule** which evaluates the flux at the center of each face

$$\oiint_{\partial\Omega_P} \phi \cdot \hat{\mathbf{n}}\, dS \approx \sum_f \phi_f A_f = \sum_f \mathcal{F}_f. \tag{5.2}$$

The summation in the equation 5.2 can be written explicitly for a cuboid CV as

$$\sum_f \phi_f A_f = \phi_e A_e - \phi_w A_w + \phi_n A_n - \phi_s A_s + \phi_t A_t - \phi_b A_b. \tag{5.3}$$

The negative signs on the West, South and Bottom terms arise for the fact that the outward normal vectors on these faces point in the negative coordinate directions.

The governing equations are discretized on a collocated grid, meaning all quantities are computed at the CV centers. Because the evaluation of mass and momentum fluxes requires values at the CVs faces, a **Central Differencing Scheme** (CDS) is used. This scheme interpolates the center values to the faces symmetrically, allowing the solver to balance the fluxes entering and exiting each CV.

The **face-centered** transported quantity ϕ_f is approximated by linear interpolation between two neighboring CVs by

$$\phi_f \approx \frac{\phi_P + \phi_F}{2}, \tag{5.4}$$

where

- F denotes the neighboring CV center, sharing the face f with CV with center P.

5.2.1 Mesh and Grid Creation

The computational domain is first discretised into a uniform grid with $N_x N_y N_z$ total nodes where N_x, N_y, N_z is the number nodes along the length, width and height of the pipe.

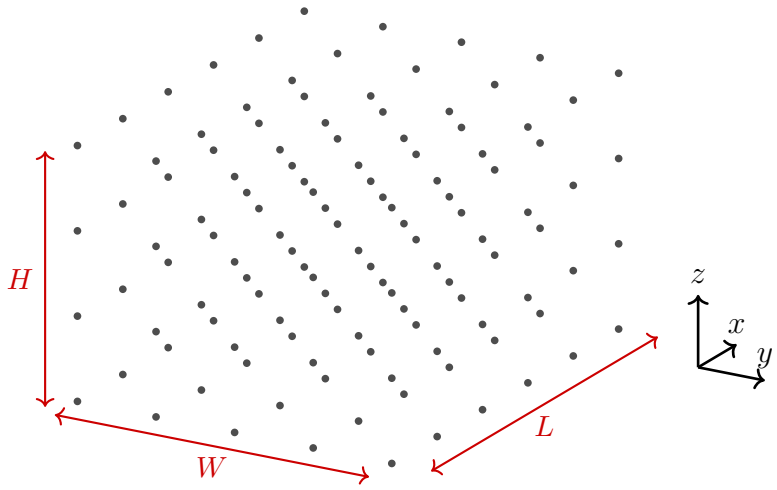


Figure 5.4: An example of a three-dimensional uniform grid defined along the dimensions of the pipe (see 4.2.1).

The node positions can be calculated as

$$x_i = i \cdot \Delta x, \qquad \Delta x = \frac{L}{N_x - 1}, \qquad i = 0, 1, \ldots, N_x - 1 \tag{5.5}$$

$$y_j = j \cdot \Delta y, \qquad \Delta y = \frac{W}{N_y - 1}, \qquad j = 0, 1, \ldots, N_y - 1 \tag{5.6}$$

$$z_k = k \cdot \Delta z, \qquad \Delta z = \frac{H}{N_z - 1}, \qquad k = 0, 1, \ldots, N_z - 1. \tag{5.7}$$

The grid introduced above provides a discrete set of node positions (x_i, y_j, z_k) that span the computational domain. However, the finite volume method requires the domain to be partitioned into a finite number of non-overlapping CVs over which the integral form of the governing equations is applied. It is therefore necessary to extend the grid into a mesh by associating each node with a surrounding control volume. Each node (x_i, y_j, z_k) is defined as the center of a cuboid CV. The faces of each CV are located midway between adjacent nodes and their positions can be calculated as

$$x_{i+1/2} := \frac{x_i + x_{i+1}}{2} = x_i + \frac{\Delta x}{2}, \qquad i = 0, 1, \ldots, N_x - 1 \tag{5.8}$$

$$x_{i-1/2} := \frac{x_{i-1} + x_i}{2} = x_i - \frac{\Delta x}{2}, \qquad i = 1, 2, \ldots, N_x \tag{5.9}$$

$$y_{j+1/2} := \frac{y_j + y_{j+1}}{2} = y_j + \frac{\Delta y}{2}, \qquad j = 0, 1, \ldots, N_y - 1 \tag{5.10}$$

$$y_{j-1/2} := \frac{y_{j-1} + y_j}{2} = y_j - \frac{\Delta y}{2}, \qquad j = 1, 2, \ldots, N_y \tag{5.11}$$

$$z_{k+1/2} := \frac{z_k + z_{k+1}}{2} = z_k + \frac{\Delta z}{2}, \qquad k = 0, 1, \ldots, N_z - 1 \tag{5.12}$$

$$z_{k-1/2} := \frac{z_{k-1} + z_k}{2} = z_k - \frac{\Delta z}{2}, \qquad k = 1, 2, \ldots, N_z. \tag{5.13}$$

The volume of each CV is $V = \Delta x \cdot \Delta y \cdot \Delta z$.
The areas of the faces are

- **East (e) and West (w):**
  Faces normal to the x-axis. The area of these faces is $A_e = A_w = \Delta y \Delta z$.
- **North (n) and South (s):**
  Faces normal to the y-axis. The area of these faces is $A_n = A_s = \Delta x \Delta z$.
- **Top (t) and Bottom (b):**
  Faces normal to the z-axis. The area of these faces is $A_t = A_b = \Delta x \Delta y$.

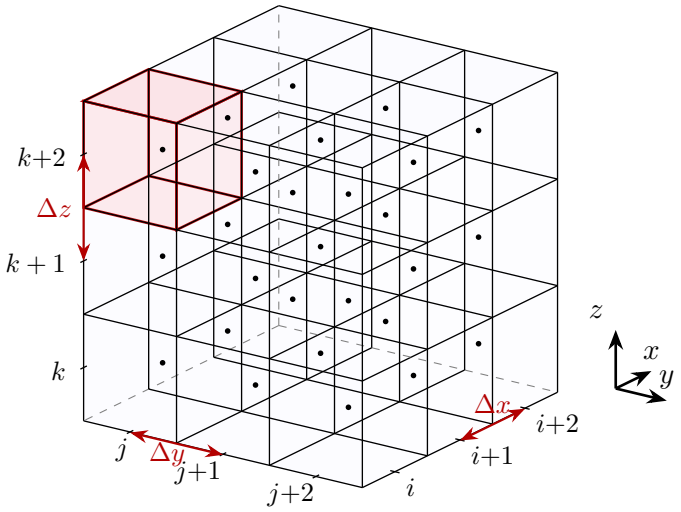


Figure 5.5: An example of a three-dimensional structured mesh composed of uniformly sized cuboid CVs. Index labels $i, i{+}1, i{+}2$ run along the x-axis, $j, j{+}1, j{+}2$ along the y-axis, and $k, k{+}1, k{+}2$ along the z-axis. Highlighted in red is an example of a CV and the dots represent the center of each CV, the dimensions of a cuboid CV is $\Delta x, \Delta y, \Delta z$.

Boundary Control Volumes and Ghost Cells

The collocated mesh defined in the previous section places the nodes at the center of each CV. However, at the boundaries of the domain, the faces of the outermost CVs coincide with the physical boundary (e.g. a wall, inlet, or outlet), while no node exists beyond it. This creates the need to handle differently the boundary CVs, in order to enforce the boundary conditions. A solution to this is to modify the discretized equations to treat the boundary CVs in a special way. Another solution, that makes computer implementation much more straightforward is to create a layer of "fake" CVs (along the boundary CVs) also known as **ghost cells** and enforce the boundary conditions there, while solving the governing equations at the centers of the boundary CVs. In other words, the solver does not solve the governing equations at the center of the ghost cells and only uses the faces of the ghost cells.

Based on the approximation order of the interpolation scheme (CDS) that is used for this simulation, one layer of ghost cells are enough for every boundary (wall, inlet, outlet), therefore the total nodes of the grid is

$$(N_x + 2) \times (N_y + 2) \times (N_z + 2) \tag{5.14}$$

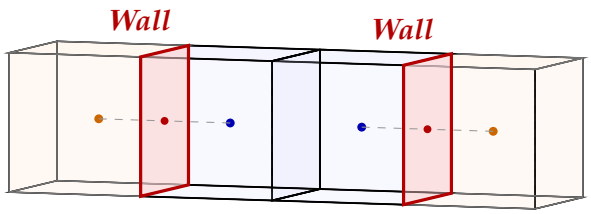


Figure 5.6: An example of a strip of CVs in three-dimensional structured mesh composed of uniformly sized cuboid CVs. The orange CVs represent the ghost cells, that are outside of the boundaries of the physical domain (represented by the red walls). The blue CVs in the middle are inside the physical domain.

5.2.2 Integration of the Governing Equations

Since FVM works primarily with volumes, the next natural step is to integrate every term of the governing equations (N–S) over a CV and apply the divergence theorem (see 2.8) to obtain the fluxes at the faces of the CV.
Integrating the dimensionless conservative N–S over Ω_P yields

$$\iiint_{\Omega_P} \nabla \cdot \mathbf{u} \; d\Omega = 0, \tag{5.15}$$

$$\iiint_{\Omega_P} \nabla \cdot (\mathbf{uu}) \; d\Omega = -\iiint_{\Omega_P} \nabla p \; d\Omega + \frac{1}{Re} \iiint_{\Omega_P} \nabla^2 \mathbf{u} \; d\Omega. \tag{5.16}$$

By applying the divergence theorem to the previous equations yields the **Integral Form** of the dimensionless conservative N–S

$$\oiint_{\partial\Omega_P} \mathbf{u} \cdot \hat{\mathbf{n}} \; dS = 0, \tag{5.17}$$

$$\oiint_{\partial\Omega_P} (\mathbf{uu}) \cdot \hat{\mathbf{n}} \, dS = -\oiint_{\partial\Omega_P} p\hat{\mathbf{n}} \, dS + \frac{1}{Re} \oiint_{\partial\Omega_P} \nabla \mathbf{u} \cdot \hat{\mathbf{n}} \, dS. \tag{5.18}$$

5.2.3 Discretization: Partial Differential Equations to Algebraic Equations

Applying the midpoint rule to each face and reconstructing the face-center values via CDS, the surface integrals are replaced by sums of discrete flux contributions over the six faces of the CV with center at (i, j, k), transforming the integral equations (presented at 5.18) into a set algebraic equations at each node.

The discretized **continuity** equation

$$\frac{\Delta y \Delta z}{2}\left(u_{i+1,j,k} - u_{i-1,j,k}\right) + \frac{\Delta x \Delta z}{2}\left(v_{i,j+1,k} - v_{i,j-1,k}\right) + \frac{\Delta x \Delta y}{2}\left(w_{i,j,k+1} - w_{i,j,k-1}\right) = 0. \tag{5.19}$$

The discretized x-**momentum** equation:

$$\begin{aligned}
&\frac{\Delta y \Delta z}{2}\left(u_{i+1,j,k}^2 - u_{i-1,j,k}^2\right) + \frac{\Delta x \Delta z}{2}\left(u_{i,j+1,k}v_{i,j+1,k} - u_{i,j-1,k}v_{i,j-1,k}\right) \\
&+ \frac{\Delta x \Delta y}{2}\left(u_{i,j,k+1}w_{i,j,k+1} - u_{i,j,k-1}w_{i,j,k-1}\right) \\
&= -\Delta y \Delta z\left(p_{i+1,j,k} - p_{i,j,k}\right) + \frac{1}{Re}\left[\frac{\Delta y \Delta z}{\Delta x}\left(u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}\right)\right. \\
&\qquad \left. + \frac{\Delta x \Delta z}{\Delta y}\left(u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}\right) + \frac{\Delta x \Delta y}{\Delta z}\left(u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}\right)\right].
\end{aligned} \tag{5.20}$$

The discretized y-**momentum** equation:

$$\begin{aligned}
&\frac{\Delta y \Delta z}{2}\left(u_{i+1,j,k}v_{i+1,j,k} - u_{i-1,j,k}v_{i-1,j,k}\right) + \frac{\Delta x \Delta z}{2}\left(v_{i,j+1,k}^2 - v_{i,j-1,k}^2\right) \\
&+ \frac{\Delta x \Delta y}{2}\left(v_{i,j,k+1}w_{i,j,k+1} - v_{i,j,k-1}w_{i,j,k-1}\right) \\
&= -\Delta x \Delta z\left(p_{i,j+1,k} - p_{i,j,k}\right) + \frac{1}{Re}\left[\frac{\Delta y \Delta z}{\Delta x}\left(v_{i+1,j,k} - 2v_{i,j,k} + v_{i-1,j,k}\right)\right. \\
&\qquad \left. + \frac{\Delta x \Delta z}{\Delta y}\left(v_{i,j+1,k} - 2v_{i,j,k} + v_{i,j-1,k}\right) + \frac{\Delta x \Delta y}{\Delta z}\left(v_{i,j,k+1} - 2v_{i,j,k} + v_{i,j,k-1}\right)\right].
\end{aligned} \tag{5.21}$$

The discretized z-**momentum** equation:

$$\begin{aligned}
&\frac{\Delta y \Delta z}{2}\left(u_{i+1,j,k}w_{i+1,j,k} - u_{i-1,j,k}w_{i-1,j,k}\right) + \frac{\Delta x \Delta z}{2}\left(v_{i,j+1,k}w_{i,j+1,k} - v_{i,j-1,k}w_{i,j-1,k}\right) \\
&+ \frac{\Delta x \Delta y}{2}\left(w^2_{i,j,k+1} - w^2_{i,j,k-1}\right) \\
&= -\Delta x \Delta y \left(p_{i,j,k+1} - p_{i,j,k}\right) + \frac{1}{Re}\left[\frac{\Delta y \Delta z}{\Delta x}\left(w_{i+1,j,k} - 2w_{i,j,k} + w_{i-1,j,k}\right)\right. \\
&\qquad \left. + \frac{\Delta x \Delta z}{\Delta y}\left(w_{i,j+1,k} - 2w_{i,j,k} + w_{i,j-1,k}\right) + \frac{\Delta x \Delta y}{\Delta z}\left(w_{i,j,k+1} - 2w_{i,j,k} + w_{i,j,k-1}\right)\right].
\end{aligned} \tag{5.22}$$

These four algebraic equations are calculated at every internal node (i, j, k) for $i = 1, \ldots, N_x$, $j = 1, \ldots, N_y$, $k = 1, \ldots, N_z$, values at positions $i = 0$, $i = N_x + 1$, $j = 0$, $j = N_y + 1, k = 0$, $k = N_z + 1$ are ghost cell values prescribed by the boundary conditions (see 5.2.1).

5.3 Assembly of the System of Algebraic Equations

The discretization process results in a local system of four coupled algebraic equations (three momentum, one continuity) for each individual CV. Because the CDS creates dependencies on neighboring nodes, these local systems cannot be solved in isolation. Instead, the local equations for every CV in the domain are assembled together to construct a **Global System of Equations** consisting of exactly $4N_xN_yN_z$ algebraic equations.

5.3.1 Global Solution Vector

To manage and solve this global system, all unknown variables (u, v, w, p) at the internal nodes are collected into a single **Global Solution Vector X** defined as

$$\mathbf{X} = \begin{bmatrix} \mathbf{U} \\ \mathbf{V} \\ \mathbf{W} \\ \mathbf{P} \end{bmatrix}, \tag{5.23}$$

where:

- **U**, **V**, **W** are the velocity vectors for the x,y,z directions.

- **P** is the pressure vector.

All four vectors are of size $(N_x \times N_y \times N_z)$, with each component containing the value of that variable at a specific interior node (i, j, k).

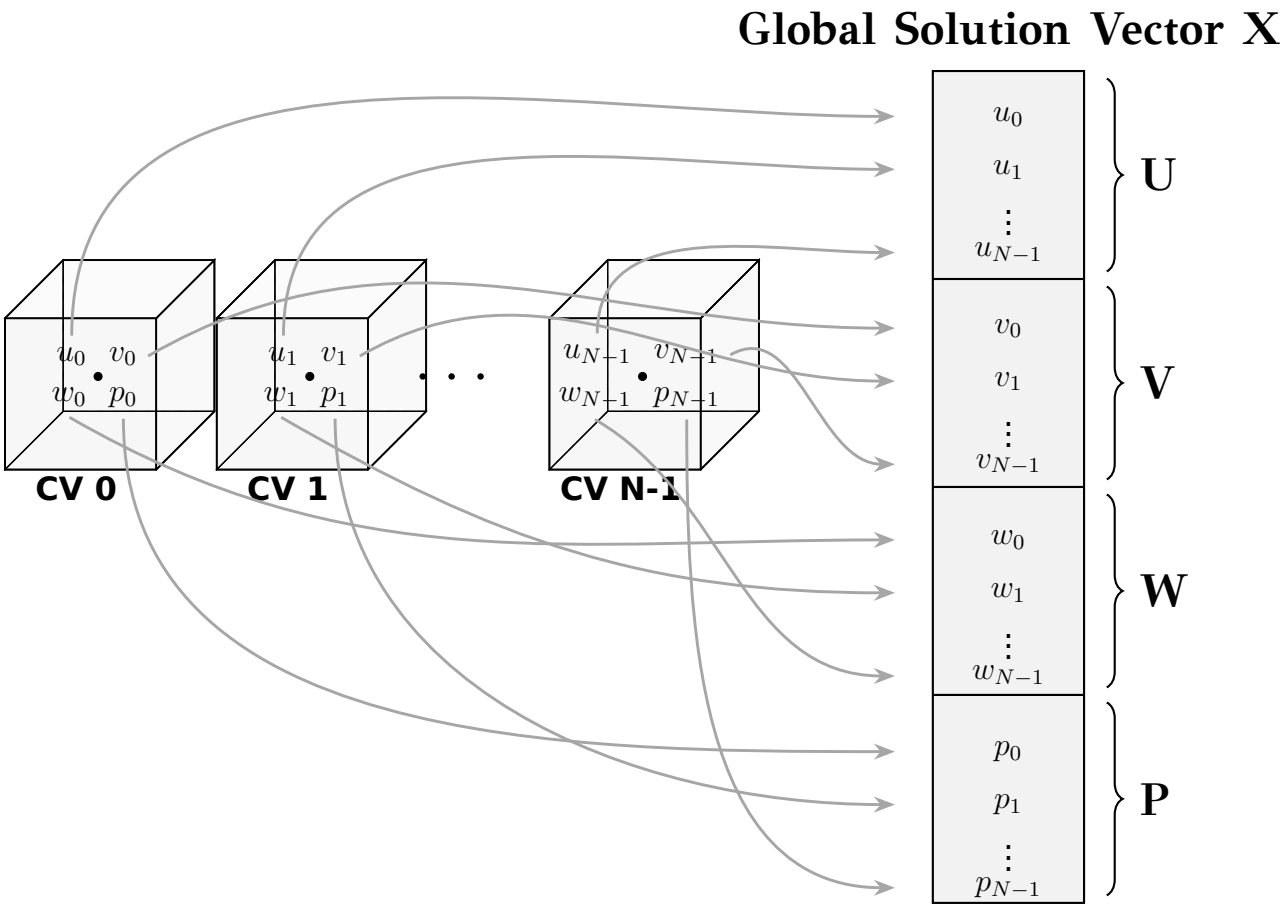


Figure 5.7: Schematic representation of the assembly process, illustrating the mapping of local variables (u, v, w, p) from individual CVs to their respective vectors (**U**, **V**, **W**, **P**) within the global solution vector **X**.

5.3.2 Residual Function

The **residual function** $\mathcal{R}(\mathbf{X})$ is comprised of four sets of scalar-valued functions, one for each unknown field:

$$\mathcal{R}(\mathbf{X}) = \begin{pmatrix} \mathcal{R}_u(\mathbf{X}) \\ \mathcal{R}_v(\mathbf{X}) \\ \mathcal{R}_w(\mathbf{X}) \\ \mathcal{R}_p(\mathbf{X}) \end{pmatrix} \tag{5.24}$$

where each component function takes the solution vector $\mathbf{X}$ as input, since each discretized equation depends simultaneously on all four unknown fields u, v, w and p, and returns the evaluation of the corresponding discretized equation at all interior nodes:

- $\mathcal{R}_u(\mathbf{X}) : \mathbb{R}^{4N_xN_yN_z} \to \mathbb{R}^{N_xN_yN_z}$ evaluates the x-momentum equation (5.20).
- $\mathcal{R}_v(\mathbf{X}) : \mathbb{R}^{4N_xN_yN_z} \to \mathbb{R}^{N_xN_yN_z}$ evaluates the y-momentum equation (5.21).
- $\mathcal{R}_w(\mathbf{X}) : \mathbb{R}^{4N_xN_yN_z} \to \mathbb{R}^{N_xN_yN_z}$ evaluates the z-momentum equation (5.22).
- $\mathcal{R}_p(\mathbf{X}) : \mathbb{R}^{4N_xN_yN_z} \to \mathbb{R}^{N_xN_yN_z}$ evaluates the continuity equation (5.19) .

The solution $\mathbf{X}$ of the nonlinear system $\mathcal{R}(\mathbf{X}) = \mathbf{0}$ is sought using iterative methods, described in the following chapter.

Chapter 6

Optimization techniques

The discretization of the governing equations presented in the previous chapter results in a system of nonlinear algebraic equations evaluated in the residual function $\mathcal{R}(\mathbf{X})$. The problem of finding the velocity components u, v, w and the pressure field p, all assembled in $\mathbf{X}$ that satisfy the N–S equations is therefore equivalent to finding the vector $\mathbf{X}^*$ that drives the residual function to zero

$$\mathbf{X}^* = \arg\min_{\mathbf{X}} \|\mathcal{R}(\mathbf{X})\|_2^2. \tag{6.1}$$

The solution $\mathbf{X}^*$ is exact when $\|\mathcal{R}(\mathbf{X}^*)\|_2 = 0$ (in practice, the iterative solver terminates when $\|\mathcal{R}(\mathbf{X}^*)\|_2 \leq \varepsilon$ for a prescribed tolerance $\varepsilon > 0$), meaning all discretized momentum and continuity equations are satisfied simultaneously at every interior node.

This is a **nonlinear least-squares problem**, since the residual $\mathcal{R}(\mathbf{X})$ is a nonlinear function of the unknowns $\mathbf{X}$. Three iterative methods for solving this class of problem are considered in this chapter. The Gradient Descent method provides the simplest iterative strategy, updating $\mathbf{X}$ in the direction of steepest descent of the objective

function. The Gauss–Newton method achieves faster convergence by approximating the Hessian from the Jacobian of $\mathcal{R}$. Finally, the Levenberg–Marquardt method combines both approaches, interpolating between gradient descent and Gauss–Newton to achieve robustness far from the solution and rapid convergence near the solution.

6.1 Jacobian of the Residual Function

The iterative methods presented in this chapter require at each iteration, knowledge of how the residual $\mathcal{R}(\mathbf{X})$ changes with respect to changes in the solution vector $\mathbf{X}$. This information is encoded in the **Jacobian matrix J** (see 2.39), which is essential for computing the update direction that drives $\mathcal{R}(\mathbf{X})$ towards zero.

The Jacobian matrix $\mathbf{J} \in \mathbb{R}^{4N_xN_yN_z \times 4N_xN_yN_z}$ is defined as the matrix of all partial derivatives of the residual function $\mathcal{R}(\mathbf{X})$ with respect to $\mathbf{X}$

$$\mathbf{J} = \frac{\partial \mathcal{R}(\mathbf{X})}{\partial \mathbf{X}} = \begin{pmatrix} \dfrac{\partial \mathcal{R}_u(\mathbf{X})}{\partial \mathbf{X}} \\ \dfrac{\partial \mathcal{R}_v(\mathbf{X})}{\partial \mathbf{X}} \\ \dfrac{\partial \mathcal{R}_w(\mathbf{X})}{\partial \mathbf{X}} \\ \dfrac{\partial \mathcal{R}_p(\mathbf{X})}{\partial \mathbf{X}} \end{pmatrix}, \tag{6.2}$$

where each block row

$$\frac{\partial \mathcal{R}_\ell(\mathbf{X})}{\partial \mathbf{X}} \in \mathbb{R}^{N_xN_yN_z \times 4N_xN_yN_z}, \qquad \ell \in \{u, v, w, p\} \tag{6.3}$$

contains the partial derivatives of the corresponding component function $\mathcal{R}_\ell(\mathbf{X})$ with respect to all entries of $\mathbf{X}$.

Since each residual entry depends only on the values at the 7-point stencil neighbors of node (i, j, k), the vast majority of entries in **J** are zero, making **J** a **sparse matrix**. This sparsity is exploited in the numerical computation of **J**, where only the nonzero entries are stored and computed, as described in ??.

6.2 Gauss-Newton method

The **Gauss–Newton** is an iterative method for solving nonlinear least-squares problems. Starting from an initial guess $\mathbf{X}_0$, it linearizes the residual function $\mathcal{R}(\mathbf{X})$ at the current iterate using the Jacobian $\mathbf{J}$, and solves the resulting linear system to obtain a solution $\mathbf{X}^*$ that reduces $\|\mathcal{R}(\mathbf{X})\|_2^2$. At each iteration n, the Gauss–Newton update is obtained by solving the following system

$$\mathbf{J}(\mathbf{X}_n)^\top \mathbf{J}(\mathbf{X}_n)\, \boldsymbol{\delta} = -\mathbf{J}(\mathbf{X}_n)^\top \mathcal{R}(\mathbf{X}_n), \tag{6.4}$$

where $\boldsymbol{\delta} \in \mathbb{R}^{4N_xN_yN_z}$ is the **update step**.
The solution vector is updated as

$$\mathbf{X}_{n+1} = \mathbf{X}_n + \boldsymbol{\delta}. \tag{6.5}$$

6.3 Gradient Descent method

Gradient descent is an iterative method that updates the solution vector $\mathbf{X}$ in the direction of the steepest descent of the $\|\mathcal{R}(\mathbf{X})\|_2^2$, which will eventually lead to local minimum. At each iteration n, $\nabla\|\mathcal{R}(\mathbf{X}_n)\|_2^2$ is computed and the solution is updated in the opposite direction, since the positive gradient points towards the steepest ascent. The gradient of $\|\mathcal{R}(\mathbf{X})\|_2^2$ is

$$\nabla\|\mathcal{R}(\mathbf{X})\|_2^2 = 2\,\mathbf{J}(\mathbf{X}_n)^\top \mathcal{R}(\mathbf{X}_n). \tag{6.6}$$

The solution vector is updated as

$$\mathbf{X}_{n+1} = \mathbf{X}_n - \alpha\,\mathbf{J}(\mathbf{X}_n)^\top \mathcal{R}(\mathbf{X}_n), \tag{6.7}$$

where $\alpha > 0$ is the **learning rate**, which controls the step size at each iteration. A large value of α can cause the method to overshoot the minimum, while a small value leads to slow convergence. The factor of 2 arising from the differentiation of the squared norm is absorbed into the learning rate α.

6.4 Levenberg-Marquardt method

The **Levenberg–Marquardt** method is an iterative algorithm that combines the robustness of gradient descent with the fast convergence of the Gauss–Newton method. Far from the solution, the method behaves like gradient descent, taking small but reliable steps towards the minimum. In the vicinity of the solution, it transitions towards the Gauss–Newton method, achieving fast convergence. This interpolation between the two methods is controlled by a damping parameter $\lambda > 0$.

The Levenberg–Marquardt update is obtained by solving the system

$$\left(\mathbf{J}(\mathbf{X}_n)^\top \mathbf{J}(\mathbf{X}_n) + \lambda\, \mathbf{I}\right) \boldsymbol{\delta} = -\mathbf{J}(\mathbf{X}_n)^\top \mathcal{R}(\mathbf{X}_n) \tag{6.8}$$

and the solution vector is updated as:

$$\mathbf{X}_{n+1} = \mathbf{X}_n + \boldsymbol{\delta} \tag{6.9}$$

where $\mathbf{I} \in \mathbb{R}^{4N_xN_yN_z \times 4N_xN_yN_z}$ is the identity matrix and $\lambda > 0$ is the **damping parameter** that controls the interpolation between the two methods. When $\lambda \to 0$ the method reduces to Gauss–Newton:

$$\mathbf{J}(\mathbf{X}_n)^\top \mathbf{J}(\mathbf{X}_n)\, \boldsymbol{\delta} = -\mathbf{J}(\mathbf{X}_n)^\top \mathcal{R}(\mathbf{X}_n) \tag{6.10}$$

and when $\lambda \to \infty$ the diagonal term $\lambda\mathbf{I}$ dominates and the update reduces to a gradient descent step:

$$\boldsymbol{\delta} \approx -\frac{1}{\lambda} \mathbf{J}(\mathbf{X}_n)^\top \mathcal{R}(\mathbf{X}_n) \tag{6.11}$$

The damping parameter λ is adapted at each iteration based on the $\|\mathcal{R}(\mathbf{X})\|_2^2$, if the update reduces the residual norm, λ is decreased to allow the method to take larger Gauss–Newton steps, else if the update increases the residual norm, λ is increased to revert towards gradient descent and take a more conservative step.

Algorithm 1 Levenberg-Marquardt algorithm

1: Compute $f = \|\mathcal{R}(\mathbf{X}_n)\|_2^2$
2: $\lambda = 10^{-3}$
3: Compute $J(\mathbf{X}_n)$
4: Assemble $J^T J$ and $J^T r$
5: Solve $\left(J^T J + \lambda I\right) \delta = -J^T \mathcal{R}(\mathbf{X}_n)$
6: Update $\mathbf{X}_{n+1} = \mathbf{X}_n + \delta$
7: Compute $f_{new} = \|\mathcal{R}(\mathbf{X}_{n+1})\|_2^2$
8: **if** $f_{new} > f$ **then** ▷ Step rejected
9: $\quad \lambda = 10\lambda$
10: $\quad$ **go to** 7
11: **else** ▷ Step accepted
12: $\quad \mathbf{X}_n \leftarrow \mathbf{X}_{n+1}$
13: $\quad \lambda = 10^{-3}$
14: $\quad f = f_{new}$
15: $\quad$ **go to** 3

Chapter 7

Implementation

7.1 High-Performance Computing

High-Performance Computing (HPC) relies on the use of super computers (clusters of powerful computers that work in conjunction with each other) to solve complex, computationally intensive problems that normal computers are unable to solve. At its core, HPC utilizes parallel processing, where massive computational tasks are divided into smaller, independent sub-tasks executed simultaneously across multiple compute nodes. These nodes typically communicate via high-speed interconnects within either shared or distributed memory architectures. Furthermore, modern systems increasingly adopt heterogeneous architectures, pairing traditional central processing units (CPUs) with specialized accelerators like graphics processing units (GPUs) to maximize computational throughput and energy efficiency.

7.1.1 Graphic Processing Units and Computational Fluid Dynamics

GPUs were originally designed for rendering graphics in real-time applications, but their massively parallel architecture has made them increasingly attractive for general-purpose scientific computing. Unlike a CPU, which typically consists of a small

number of powerful cores optimised for sequential tasks, a GPU contains thousands of smaller cores designed to execute many operations simultaneously. This makes GPUs particularly well suited for the type of large-scale numerical computations that arise in CFD, where the same operation must be applied to thousands or millions of unknowns at every iteration.

In the context of CFD, the most computationally intensive operations are the evaluation of the residual function, the computation of the Jacobian matrix, and the solution of the resulting system, all of which involve large number of numerical operations that can be parallelized effectively on a GPU. The CUDA programming model, developed by NVIDIA, provides a framework for executing such operations directly on the GPU. In this study, the entire solution pipeline, from the evaluation of the governing equations to the solution of the sparse linear system at each Levenberg–Marquardt iteration, is executed on the GPU, taking advantage of the CUDA libraries including cuDSS, cuBLAS for dense linear algebra and cuSPARSE for sparse matrix operations.

7.2 Technical Details

The solver is implemented in C++17, taking advantage of modern language features such as structured bindings and the standard template library for efficient memory management (`std::vector` and move semantics). GPU acceleration is achieved through CUDA 13.1, which takes advantage of the latest features of NVIDIA GPUs. On the CPU side, shared-memory parallelism is exploited using OpenMP, enabling multi-threaded execution of operations that are performed on the host. The combination of OpenMP for CPU parallelism and CUDA for GPU acceleration ensures that computational resources are utilised efficiently at both levels of the hardware hierarchy.

Visualization of the simulation results are handled in Python, which provides a rich and accessible environment for analyzing the output of the solver. The results produced by the C++ solver are written to output files and subsequently read by Python scripts.

Component	Tool / Version
Programming language	C++17
GPU programming model	CUDA 13.1
CPU parallelism	OpenMP
Sparse direct solver	NVIDIA cuDSS
Dense linear algebra	NVIDIA cuBLAS
Sparse linear algebra	NVIDIA cuSPARSE
Visualization	Python matplotlib
Documentation	Doxygen
Version control	GitHub

Table 7.1: Software stack used in the implementation.

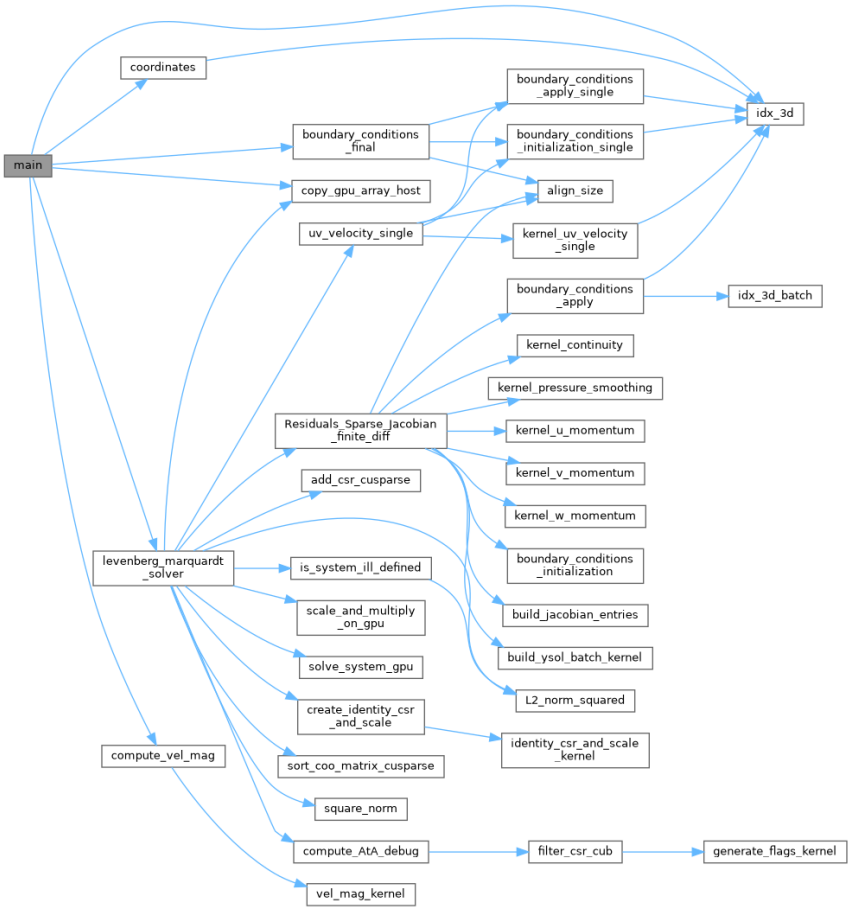


Figure 7.1: This diagram illustrates the function dependency and execution flow of the C++ part of the implementation.

7.3 Conventions and Architecture Decisions

In the design and implementation phase, all architectural decisions are guided by improving the performance. To maximize memory throughput, data structures are flattened into one-dimensional arrays. This approach takes full advantage of contiguous memory blocks and efficient memory coalescing. Realizing that data transfers between the CPU and the GPU act as a bottleneck therefore are avoided as much as possible, also this overhead is mitigated by utilizing pinned memory. From the GPU perspective, several specific conventions are applied to achieve better performance, the source code is compiled for the specific target GPU architecture (SM-89 for example), utilizing fast math compilation flags and 64-bit precision. Finally, execution efficiency is further improved by implementing overlapping kernel execution and deliberately reducing register pressure through the use of precomputed values.

Bibliography

[1] F. Moukalled, L. Mangani, and M. Darwish, *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab*, ser. Fluid Mechanics and Its Applications. Springer International Publishing, vol. 113. [Online]. Available: https://link.springer.com/10.1007/978-3-319-16874-6

[2] M. Fratarcangeli, D. Bradley, A. Gruber, G. Zoss, and T. Beeler, "Fast nonlinear least squares optimization of large-scale semi-sparse problems," *Computer Graphics Forum*, vol. 39, no. 2, p. 247–259, May 2020.

[3] K. Madsen, H. B. Nielsen, and O. Tingleff, "Methods for non-linear least squares problems (2nd ed.)," 2004. [Online]. Available: https://api.semanticscholar.org/CorpusID:64217935

[4] H. K. Versteeg and W. Malalasekera, *An introduction to computational fluid dynamics: the finite volume method*, 2nd ed. Harlow: Pearson/Prentice Hall, 2007.

[5] NVIDIA Corporation, *cuSPARSE Library*, NVIDIA, 2026, version 13.1. [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUSPARSE_Library.pdf

[6] G. Strang, *Linear Algebra and Its Applications*, 4th ed. Brooks/Cole, Cengage Learning, 2006.

[7] J. E. Marsden and A. Tromba, *Vector Calculus*, 6th ed. W. H. Freeman, Dec. 2011.

[8] S. Katsoudas, S. Malatos, A. Raptis, M. Matsagkas, A. Giannoukas, and M. Xenos, "Blood flow simulation in bifurcating arteries: A multiscale approach after fenestrated and branched endovascular aneurysm repair," *Mathematics*, vol. 13, no. 9, p. 1362, Apr. 2025.

[9] E. Lauga, *Fluid Mechanics: A Very Short Introduction*, ser. Very Short Introductions. Oxford University Press, 2022.

[10] P. Chassaing, *Fundamentals of Fluid Mechanics: For Scientists and Engineers*. Cham: Springer International Publishing, 2022. [Online]. Available: https://link.springer.com/10.1007/978-3-031-10086-4

[11] NVIDIA Corporation, *cuBLAS Library*, 2026, version 13.1. [Online]. Available: https://docs.nvidia.com/cuda/cublas/

[12] ——, *CUDA C++ Best Practices Guide*, 2026. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/

[13] Embry-Riddle Aeronautical University, 2022. [Online]. Available: http://dx.doi.org/10.15394/eaglepub.2022.1066.n16

Appendix A

Supplementary Material

A.1 Conservation of Mass (Continuity Equation)-Integral Form

Determinant of a Matrix

The **determinant** is a scalar function $\det(\mathbf{A}) \in \mathbb{R}$ defined for every square matrix $\mathbf{A} = [a_{ij}]$.

For a 2×2 matrix the determinant is

$$\det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11}\, a_{22} - a_{12}\, a_{21}. \tag{A.1}$$

For a 3×3 matrix the determinant is computed by cofactor expansion along the first row:

$$\det \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{12}(a_{21}a_{33} - a_{23}a_{31}) + a_{13}(a_{21}a_{32} - a_{22}a_{31}). \tag{A.2}$$

For an $n \times n$ matrix, there are many equivalent ways to define the determinant. The most common and simple is **Leibniz Formula** (Although not computationally efficient).

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{\sigma(i),i} \tag{A.3}$$

where the sum runs over all $n!$ permutations σ of the set $\{1, 2, \ldots, n\}$, and $\operatorname{sgn}(\sigma) = \pm 1$ is the sign (or parity) of the permutation σ. The sign is $+1$ for even permutations (an even number of transpositions) and -1 for odd permutations.

For $n = 2$ this reproduces equation (A.1), and for $n = 3$ it produces the six-term expression equivalent to equation (A.2).

A.0.1 Eigenvectors and Eigenvalues

A non-zero vector $\mathbf{v} \in \mathbb{R}^n$ is called an **eigenvector** of a square matrix $\mathbf{A}$, if multiplying it by $\mathbf{A}$ produces a scalar multiple of itself:

$$\mathbf{A}\,\mathbf{v} = \lambda\,\mathbf{v}, \qquad \mathbf{v} \neq \mathbf{0}. \tag{A.4}$$

The scalar $\lambda \in \mathbb{R}$ associated with $\mathbf{v}$ is called the corresponding **eigenvalue**.

The eigenvalues of a square matrix $\mathbf{A}$ can be found by solving this equation:

$$\mathbf{A}\,\mathbf{v} - \lambda\,\mathbf{v} = \mathbf{0} \implies (\mathbf{A} - \lambda\,\mathbf{I})\,\mathbf{v} = \mathbf{0}, \tag{A.5}$$

which is derived by rearranging (A.4).
Geometrically, $\mathbf{A}$ stretches or compresses $\mathbf{v}$ by a factor of λ without changing its direction (or reversing it if $\lambda < 0$).

A.0.2 Symmetric Positive-Definite Matrices

A square matrix $\mathbf{A}$ is called **symmetric positive-definite** (SPD) if it satisfies two conditions simultaneously.

First, it must be symmetric and second, for every non-zero vector $\mathbf{x} \in \mathbb{R}^n$, the quadratic form must be strictly positive:

$$\mathbf{x}^\top \mathbf{A}\,\mathbf{x} > 0 \qquad \forall\,\mathbf{x} \neq \mathbf{0}, \tag{A.6}$$

where the scalar $\mathbf{x}^\top \mathbf{A}\mathbf{x}$ is called the **quadratic form** associated with $\mathbf{A}$.

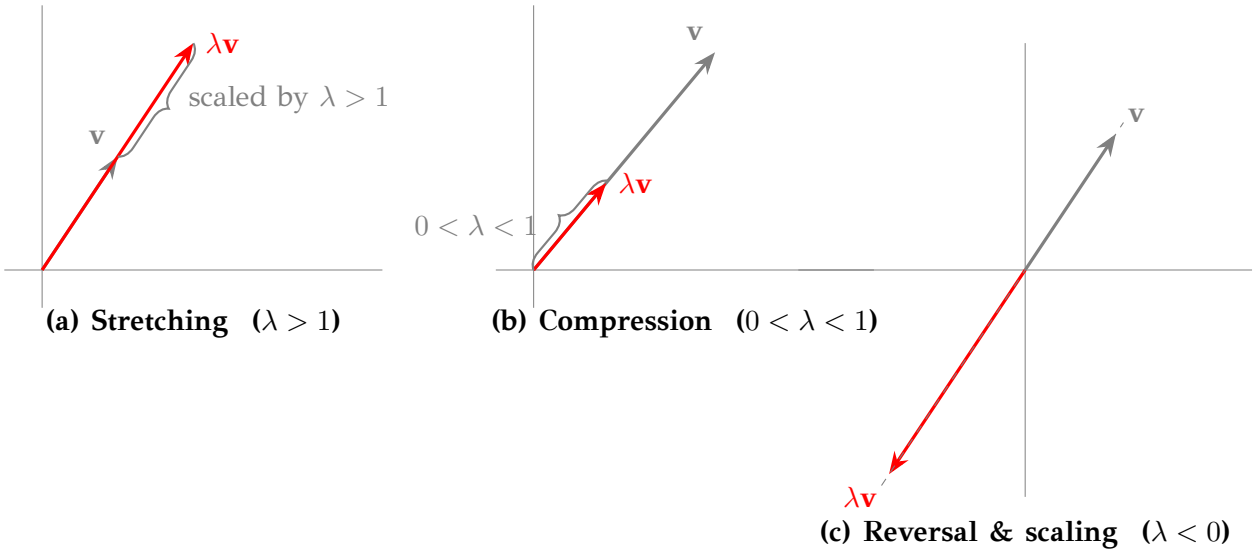


Figure A.1: Geometric effect of an eigenvalue λ on its eigenvector $\mathbf{v}$.

A.1 Conservation of Mass (Continuity Equation)-Integral Form

Consider a fixed Cuboid control volume $V \subset \Omega$ bounded by a closed surface S with outward unit normal vector $\mathbf{n}$.

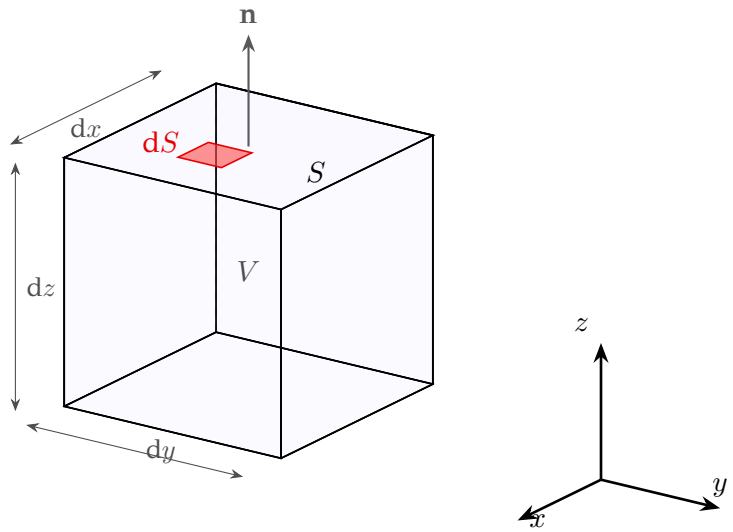


Figure A.2: Cuboid control volume V with outward unit normal vectors $\mathbf{n}$. Blue coloring represents the closed surface S. The small red rectangle represent the infinitesimal area element $\mathrm{d}S$ on the closed surface S.

The conservation of mass states that mass can neither be created nor destroyed.

The total mass contained in V is $\iiint_V \rho \, \mathrm{d}V$, where $\rho = \rho(\mathbf{x}, t)$ is the fluid density and $\mathrm{d}V = \mathrm{d}x \, \mathrm{d}y \, \mathrm{d}z$ is the infinitesimal volume element. Conservation of mass requires

that the rate of change of this mass equals the net rate of mass flowing into V through S:

$$\frac{\partial}{\partial t} \iiint_V \rho \, \mathrm{d}V \; + \; \oiint_S \rho \, \mathbf{u} \cdot \mathbf{n} \, \mathrm{d}S \; = \; 0 \tag{A.7}$$

The first term describes the time rate of accumulation of mass inside V, meaning how fast or slow the mass is changing inside V. The second term is the net mass flux leaving V through S, in simple terms how fast or slow the mass flows out of V. In order to "conserve" the mass what goes in must go out.

Equation (A.7) is the **integral form** of the continuity equation and holds for any control volume, regardless of the flow regime or fluid type.

As stated, no mass is created, destroyed, or compressed anywhere in the domain. This constraint is enforced at every iteration of the numerical solver and acts as a consistency condition on the velocity and pressure fields throughout the simulation.

A.1.1 Body Forces

Body forces are forces that act throughout the CV of the fluid without physical contact, such as gravity and electromagnetic forces and can be described as

$$\mathbf{f_b} = f_x \mathbf{i} + f_y \mathbf{j} + f_z \mathbf{k} \tag{A.8}$$

Gravity, g is a omnipresent force that acts on the axis z,

$$\mathbf{f_b} = f_x \mathbf{i} + f_y \mathbf{j} + (f_z - g) \mathbf{k}. \tag{A.9}$$

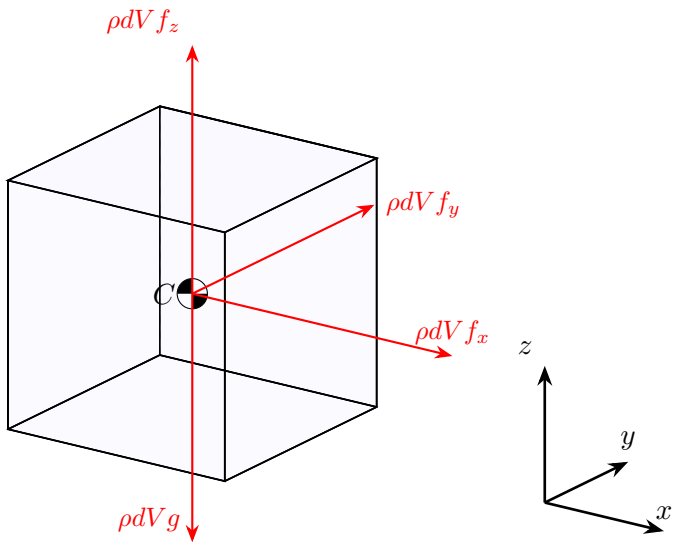


Figure A.3: Cuboid CV showing the body forces (red arrows) for each axis. Point C represents the center of mass of the CV.

A.1.2 Pressure Term

Applying the midpoint rule to the pressure surface integral over the east and west faces of Ω_P gives:

$$\oiint_{\partial\Omega_P} p\, n_x\, dS \approx p_e\, \Delta y \Delta z - p_w\, \Delta y \Delta z \tag{A.10}$$

Reconstructing the face values via CDS:

$$p_e \approx \frac{p_{i+1,j,k} + p_{i,j,k}}{2}, \qquad p_w \approx \frac{p_{i,j,k} + p_{i-1,j,k}}{2} \tag{A.11}$$

yields:

$$\left(\frac{p_{i+1,j,k} + p_{i,j,k}}{2} - \frac{p_{i,j,k} + p_{i-1,j,k}}{2}\right) \Delta y \Delta z = \frac{p_{i+1,j,k} - p_{i-1,j,k}}{2}\, \Delta y \Delta z \tag{A.12}$$

The center value $p_{i,j,k}$ cancels entirely from the pressure gradient, decoupling the pressure at node (i, j, k) from the surrounding velocity field and producing a checkerboard instability on the collocated mesh. To maintain pressure-velocity coupling, CDS is therefore replaced by a one-sided difference evaluated between the east face and the CV center

$$\oiint_{\partial\Omega_P} p\, n_x\, dS \approx (p_{i+1,j,k} - p_{i,j,k})\, \Delta y \Delta z \tag{A.13}$$

and analogously for the y- and z-directions:

$$\oiint_{\partial\Omega_P} p\, n_y\, dS \approx (p_{i,j+1,k} - p_{i,j,k})\, \Delta x \Delta z, \qquad \oiint_{\partial\Omega_P} p\, n_z\, dS \approx (p_{i,j,k+1} - p_{i,j,k})\, \Delta x \Delta y \tag{A.14}$$