Practical Introduction to Computational Fluid Dynamics

Khalid M. Saqr, Ph.D.

Course Notes

Second Edition

January 2017

First Edition, November 2015

Second Edition, January 2017

Copyright © 2017 by Khalid M. Saqr

All rights reserved. Any commercial reproduction or use of this publication is strictly prohibited. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the author, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law. For any permissions of this publication, requests and inquiries should be sent to the author's email on k.saqr@aast.edu or by post to the following address:

Dr. Khalid M. Saqr
Mechanical Engineering Dept.,
College of Engineering and Technology
Arab Academy for Science, Technology and Maritime Transport
P.O. Box 1029, Abu Qir, Alexandria – EGYPT

Disclaimer of Liability

The author specifically DISCLAIM LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES and assumes no responsibility or liability for any loss or damage suffered by any person as a result of the use or misuse of any of the information or content provided in this publication. The author assumes or undertakes NO LIABILITY for any loss or damage suffered as a result of the use, misuse or reliance on the information and content of this publication.

ANSYS®, Fluent® are registered trademarks of ANSYS Inc., Pennsylvania, United States.
OpenFOAM® is a registered trademark of OpenCFD, U.K.

This is a special edition digitally produced by author permission for the High-Speed Reacting Flow Laboratory (HiREF) of Universiti Teknologi Malaysia.

Table of Contents

Table of Contents 4

Preface 7

Governing equations of fluid flow 8

1.1. Velocity vector 8

1.2. Acceleration and the substantial derivative 8

1.3. Governing Equations of Fluid Flow 9

1.3.1. Continuity equation 10

1.3.2. Momentum equation 10

1.3.3. Energy equation 14

The finite Volume Method 16

1.4. Gauss (divergence) theorem 17

1.5. Stokes theorem 18

1.6. The finite volume method 18

1.6.1. Grid generation 19

1.6.2. Discretization 19

1.6.3. Preparation of the linear equations system 20

1.6.4. Application of boundary conditions 20

Elements of a CFD solver 23

1.7. Computational grid 23

1.7.1. Structured grids 23

1.7.2. Unstructured grids 23

1.8. Numerical discretization schemes 24

1.8.1. First order upwind scheme 26

1.8.2. Second order upwind scheme 26

1.8.3. Central differencing scheme 27

1.8.4. Power law scheme 27

1.8.5. QUICK Scheme 27

1.8.6. Accuracy of discretization schemes 27

1.9. Linear equation solution methods 28

1.9.1. Direct Methods 29

1.9.1.1. Gauss Elimination 29

1.9.1.2. LU Decomposition 30

1.10. Iterative methods 31

1.10.1. Basic Concept 31

1.10.2. The Gauss-Seidel point-by-point method 32
1.10.3. Incomplete LU Decomposition: Stone's Method 32
1.10.4. Conjugate Gradient Method 35
1.10.5. Biconjugate Gradients and CGSTAB 37
1.11. Pressure-Velocity coupling 39
1.12. Under relaxation of flow field variables 39
1.13. Solution convergence 40
Introduction to turbulence modeling 41
1.14. Properties of turbulence 41
1.14.1. Randomness and fluctuations 41
1.14.2. Nonlinearity 41
1.14.3. Diffusion 41
1.14.4. Vorticity/eddies/energy cascade 42
1.14.5. Dissipation 42
1.14.6. Mean flow features 42
1.15. The Reynolds Averaged Navier-Stokes (RANS) equation 43
1.15.1. Time averaging 43
1.15.2. Ensemble averaging 43
1.15.3. Spatial averaging 44
1.15.4. Derivation of the RANS equation 44
1.16. Boussinesq eddy viscosity assumption 45
1.17. Most common eddy viscosity turbulence models 45
1.17.1. The standard $k-\varepsilon$ model 45
1.17.2. The realizable $k-\varepsilon$ model 45
1.17.3. The standard $k-\omega$ model 46
Verification and Validation in CFD 48
Bibliography 50

List of Figures

Figure 1. Representation of the velocity vector and its components in Cartesian coordinates 8
Figure 2. A moving fluid element showing the velocity as a function of space and time 9
Figure 3. Classification of forces acting on a moving fluid element 11
Figure 4. Surface forces acting on a moving fluid element due to a force in **x** direction. 12
Figure 5. Schematic diagram for the components of a CFD code 16
Figure 6. Computational domain for submerged jet flow 17
Figure 7. Open surface S with a bounding surface C 18
Figure 8 Control volumes for a one dimensional, steady-state diffusion problem 19
Figure 9. Computational domain specification for the 1-D steady diffusion problem 19
Figure 10. Schematic of the 1-D steady heat conduction in a rod 21
Figure 11. Example of a 2D, structured, non-orthogonal grid, designed for calculation of flow in a symmetry segment of a staggered tube bank 23
Figure 12. Example of a 2D unstructured grid 24
Figure 13. Schematic of the computational domain of a steady 2-D diffusion convection problem. The upper figure shows the cell indices and the lower shows the cell size and neighbor cells indices. 25
Figure 14. Structure of the matrix for a five-point computational molecule (nonzero entries in the coefficient matrix on five diagonals are shaded; each horizontal set of boxes corresponds to one grid line). 28
Figure 15. Schematic presentation of the matrices L and U and the product matrix M; diagonals of M not found in A are shown by dashed lines 33
Figure 16. Flowchart of the SIMPLE algorithm loop 39
Figure 17. Stationary and non-stationary time series of a turbulent flow 43
Figure 18. An ensemble of the function u(t). 43
Figure 19. Sources of error in CFD 48
Figure 20. Verification and Validation process flowchart 49

Preface

Computational Fluid Dynamics (CFD) has become an emerging field in engineering since the early 1980s. The outbreak of personal computers during that era has revolutionized engineering practice in many industries and started a technology race between software companies to develop engineering analysis and simulation software for a wide range of applications. Fluid dynamics is crucially important to almost all engineering applications; from automotive and process engineering to power generation and defense industries. Hence, it was natural for CFD to be within the core of the global quest to develop and possess accurate and reliable simulation software As a result, CFD has become a very important analysis tool, as well as area of research, in academia. Since it involves mathematical modeling, numerical analysis, physics and scientific computing, CFD is an ideal example of interdisciplinary research areas. Mathematicians, physicists, engineers and computer scientists found great passion in developing new CFD tools and improve the state of the art codes. However, as it is common in similar interdisciplinary fields, the boundaries between these specializations are indefinite, dynamic, and sometimes vague. Mechanical engineers who wish to make use of the powerful CFD tools today must leave their comfort zone of empirical and closed form correlations to get into differential calculus, numerical analysis, and scientific computing. Mathematicians who aim at improving the state of the art discretization schemes must understand the physics of turbulent flow and chemical reactions. Computer scientists who wish to employ the increasing power of Graphical Processing Units in CFD should understand the nature of initial-boundary problems encountered in CFD to know the type of matrices they yield. As a result, text and reference books of CFD attempt to demonstrate such interdisciplinary nature of the field. Unlike the classic books of the 1980s and 1990s, modern CFD books do discuss state of the art methods implemented in general purpose codes.

These notes is an attempt to provide mechanical engineers with a practical resource of the fundamentals of CFD to enable complete understanding of the most common CFD methods used in mechanical engineering applications today. The notes are interdisciplinary of course, however, only the boundaries that interest junior mechanical engineers were followed. The first two chapters provide the mathematical foundations of differential analysis of fluid flow and its governing equations. The third chapter sheds the light on the most commonly used discretization method in today's general purpose CFD codes; the finite volume method. Chapter four provides the reader with the essential elements of a CFD solver, such that their suitability for the problems usually encountered in mechanical engineering practice is understood. Chapter five introduces the reader to turbulence modeling, and focuses on the most commonly used models which are eddy viscosity models. In chapter six, the reader learns about the reliability assessment of CFD simulations through verification and validation practices. Chapter seven gives a brief introduction about modeling of combustion systems using CFD, and chapter eight provides practical examples using ANSYS Fluent®.

Khalid M. Saqr, Ph.D.
Alexandria, 2015
Alexandria, 2017

Governing equations of fluid flow

1.1. Velocity vector

The motion of fluids is much more complex than such of solid bodies. Fluids in motion always change their shape, hence the representation of flow field variables (such as velocity and pressure) must be in field functions; whither these functions are scalars (such as pressure and temperature), vectors (such as velocity) or tensors (such as strain rate). The most important flow field function is the *velocity vector field*, which assumes different values and directions along different points in the flow, and from one instance of time to another. Therefore, assuming a moving fluid in generalized Cartesian coordinates, as in figure (1) its velocity vector $\vec{V}(x, y, z, t)$ always has three components $\vec{u}$, $\vec{v}$ and $\vec{w}$ in **x**, **y** and **z** directions, respectively. Giving the three dimensional nature of an arbitrary fluid motion, each of the velocity vector components is a function of space and time, so that the velocity vector can be expressed in terms of its components as:

$$\vec{V} = \vec{u}(x, y, x, t)\hat{\imath} + \vec{v}(x, y, x, t)\hat{\jmath} + \vec{w}(x, y, x, t)\hat{k}. \qquad (1.1)$$

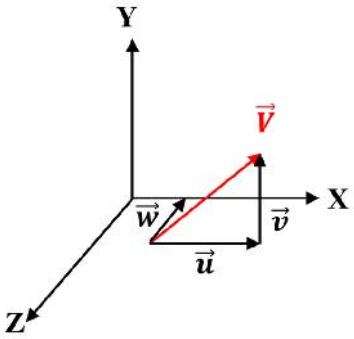


Figure 1. Representation of the velocity vector and its components in Cartesian coordinates

1.2. Acceleration and the substantial derivative

In general, fluid flow is a *time-dependent* phenomena. If we consider the motion of a single fluid particle from one point to another, as in figure 2, the velocity varies due to two reasons. The first reason is the change of position from point (1) to point (2). The second reason is the passage of time from t_1 to t_2. Given the fact that there is an enormous number of fluid particles in any flow, one can imagine how the velocity *vector field* varies. Hence, the concept of *acceleration* here is different from such in the motion of solid objects, where there is no mass transfer within the solid object itself during motion, and where velocity changes only due to the passage of time. Therefore, a distinct expression of fluid acceleration must be introduced. This can be done by differentiating the velocity vector, as given in equation (1.1) with respect to time, as following:

$$\frac{\partial}{\partial}\left(\vec{V}\right) = \frac{\partial \vec{V}}{\partial} + \frac{\partial \vec{V}}{\partial}\frac{\partial x}{\partial} + \frac{\partial \vec{V}}{\partial}\frac{\partial y}{\partial} + \frac{\partial \vec{V}}{\partial}\frac{\partial z}{\partial} \qquad (1.2)$$

$$\frac{\partial}{\partial}\left(\vec{\boldsymbol{V}}\right) = \frac{\partial \vec{\boldsymbol{V}}}{\partial} + \vec{\boldsymbol{u}}\frac{\partial \vec{\boldsymbol{V}}}{\partial} + \vec{\boldsymbol{v}}\frac{\partial \vec{\boldsymbol{V}}}{\partial} + \vec{\boldsymbol{w}}\frac{\partial \vec{\boldsymbol{V}}}{\partial} \quad (1.3)$$

The left hand side of the equation represent the acceleration field. Simplifying the vector expression by removing the above arrow sign, the components of acceleration in **x**, **y** and **z** directions, can be expressed respectively as:

$$a_x = \frac{\partial}{\partial} + u\frac{\partial}{\partial} + v\frac{\partial}{\partial} + w\frac{\partial}{\partial} \quad (1.4)$$

$$a_y = \frac{\partial}{\partial} + u\frac{\partial}{\partial} + v\frac{\partial}{\partial} + w\frac{\partial}{\partial} \quad (1.5)$$

$$a_z = \frac{\partial}{\partial} + u\frac{\partial}{\partial} + v\frac{\partial}{\partial} + w\frac{\partial}{\partial} \quad (1.6)$$

By introducing the Cartesian differential operator $\vec{\nabla} = \frac{\partial}{\partial}\hat{\imath} + \frac{\partial}{\partial}\hat{\jmath} + \frac{\partial}{\partial}\hat{k}$ to the notation of equation (1.4 – 1.6), equation (1.3) can be written as:

$$\frac{D\boldsymbol{V}}{D} \equiv \frac{\partial \boldsymbol{V}}{\partial} + \boldsymbol{V}(\nabla \cdot \boldsymbol{V}) \quad (1.7)$$

The first term of the right hand side is called *local* acceleration which is the time rate of change of velocity. The second term of the right hand side is called *convective acceleration* which is the spatial rate of change of velocity. The substantial derivative is

$$\frac{D}{D} \equiv \frac{\partial}{\partial} + (\nabla \cdot V) \quad (1.8)$$

where the term $(\nabla \cdot V)$ is the time rate of change of the volume of a moving fluid element per unit volume, which is called the *velocity divergence*.

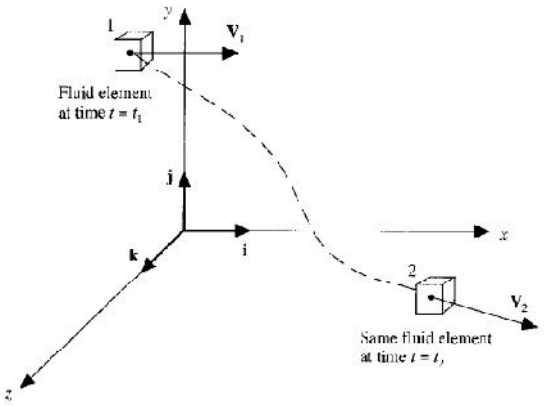


Figure 2. A moving fluid element showing the velocity as a function of space and time

1.3. Governing Equations of Fluid Flow

The term *governing equations* resembles a system of differential equations derived from the basic conservation principles which *govern* the philosophical framework of classical continuum mechanics. These principles are the conservation of mass, momentum and energy. When these principles are applied to a stationary frame of reference, the resulting form of governing equations are called *conservation* form. On the other hand,

these equations are called *non-conservation* form when the conservation principles are applied to a moving frame of reference. Let's begin by applying the conservation of mass principle to derive the continuity equation of a fluid flow.

1.3.1. Continuity equation

Assume a moving fluid element with an infinitesimally small mass δ and variable volume δ :

$$\delta = \rho\,\delta \tag{1.9}$$

since mass is conserved, then $\frac{D(\delta)}{D} = 0$, hence, by substituting and expanding in (1.9), we get:

$$\frac{D(\delta)}{D} = \frac{D(\rho\delta)}{D} = \rho\frac{D(\delta)}{D} + \delta\,\frac{D}{D} = 0 \tag{1.10}$$

dividing both sides by δ , and rearranging, we get:

$$\frac{D}{D} + \rho\left[\frac{1}{\delta}\frac{D(\delta)}{D}\right] = 0 \tag{1.11}$$

since the second term on the left hand side represent the time rate of change of volume per unit volume, equation (1.11) can be written as:

$$\frac{D}{D} + \rho(\nabla \cdot V) = 0 \tag{1.12}$$

Equation (1.12) is called the continuity equation for unsteady compressible flow. This is a *partial differential* form of the equation, which can be manipulated into *integral* form. Equation (1.12) is in *non-conservation* form, and can be written in *conservation* form as:

$$\frac{\partial}{\partial} + \nabla(\rho) = 0 \tag{1.13}$$

1.3.2. Momentum equation

The momentum equation is derived by applying the conservation of momentum principle, which is also known as Newton's second law, to the fluid in motion. Newton's second law dictates that the rate of change of linear momentum is equal to sum of forces acting on/within the moving fluid. Momentum is the product of mass of a fluid element and its velocity, hence, it is a function of space and time. Applying Newton's second law on a moving fluid element, as such shown in figure 2, we get: $m \cdot a = \sum F$, where the left hand side represents the time rate of change of linear moment $\left(\frac{d}{d}(m.V)\right)$, and the right hand side is the sum *(i.e. resultant)* of forces acting on the fluid element. The mass of the fluid element can be expressed as:

$$m = \rho\,\delta = \rho = \rho \tag{1.14}$$

hence, the left hand side of the momentum equation, for incompressible flows, can expressed by combining equations (1.7) and (1.14), to get:

$$m \cdot a = \rho\frac{D\boldsymbol{V}}{D}d = \rho\left(\frac{\partial \boldsymbol{V}}{\partial} + (\boldsymbol{V} \cdot \nabla \boldsymbol{V})\right)d \tag{1.15}$$

In order to get the sum of forces acting on a moving fluid element, we must first identify their sources. These forces can be classified to two categories; body and surface forces, as shown in figure 3. Body forces are originated outside the fluid volume, and act on all fluid elements (i.e. particles) with the same magnitude and direction, such as gravitational force or electromagnetic force. Surface forces are originated within the moving fluid, due to motion itself, and act on different fluid elements with different magnitudes and directions. They are classified to pressure force and viscous force. Pressure force occurs due to the packing (i.e. pushing) of fluid particles due to the presence of confinements. Viscous forces occur due to fluid friction. There are two mechanisms of friction; normal viscous stress and shear viscous stress. Although normal viscous stress acts in the same direction of pressure force, it has a different source and nature. Shear viscous force acts to *twist* fluid particles. Figure 4 schematically shows the different components of surface forces acting on a moving fluid element. The element assumes a cubical shape with six faces, four of which are denoted by the first letter of the cardinal directions (side faces) as well as the upper and lower faces. Here, the coordinate system is a right-hand one, meaning that the direction for velocity increase is always the positive direction of **x**, **y** and **z**.

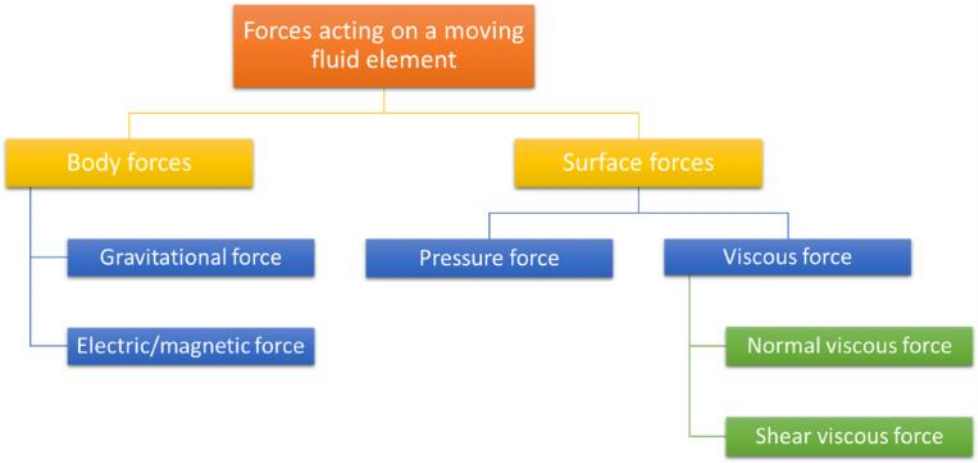


Figure 3. Classification of forces acting on a moving fluid element

Here, we recall that the units of force is Newton ($kg.m.s^{-2}$), and the units of stress is Newton per unit area or ($kg.m^{-1}.s^{-2}$). Let us assume that the body force (gravitational force in the arbitrary case) per unit mass acting on the fluid element is denoted by **F**, hence, its component in **x** direction is $\boldsymbol{f}_x$. Therefore, the total body force acting on the fluid element in **x** direction can be given as:

Body force in **x** direction = $\rho \boldsymbol{f}_x \, d$ (1.16)

While the body force components in **y** and **z** directions can be given, respectively, as:

Body force in **y** direction = $\rho \boldsymbol{f}_y \, d$ (1.17)

Body force in **z** direction = $\rho \boldsymbol{f}_x \, d$ (1.18)

As to the pressure force in the **x** direction, it acts on the faces denoted by (E) and (W). The net pressure force on the element in the **x** direction is given by:

$$\text{Net pressure force in the } \mathbf{x} \text{ direction } (f_x^p) = \left(p - \left(p + \frac{d}{d} d\right) d\right)$$

$$= -\frac{d}{d} d \qquad (1.19)$$

and the pressure forces in **y** and **z** directions, which act on the faces denoted by (U & L) and (N & S), respectively, are given similarly as:

$$f_y^p = -\frac{d}{d} d \qquad (1.20)$$

$$f_z^p = -\frac{d}{d} d \qquad (1.21)$$

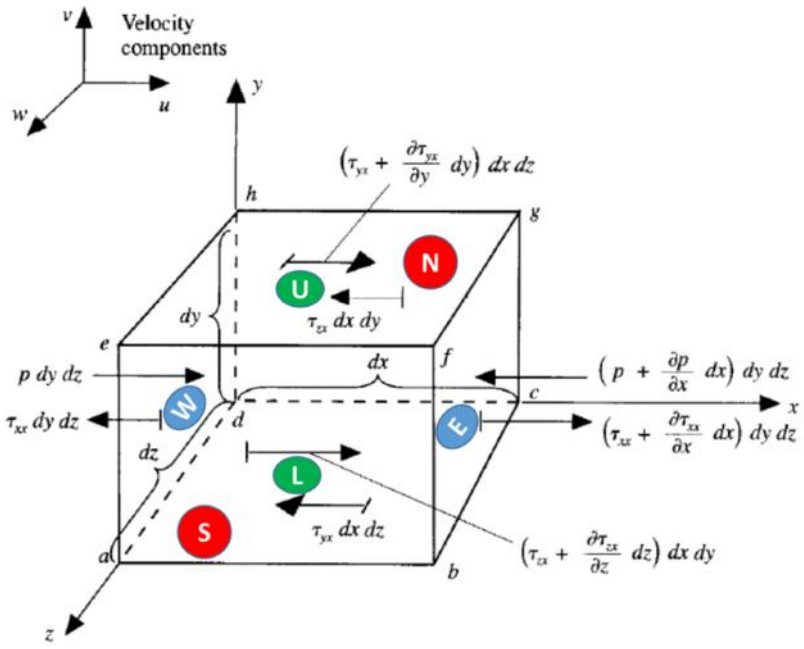


Figure 4. Surface forces acting on a moving fluid element due to a force in **x** direction.

For viscous forces and stresses, subscripts are used to denote the plane and direction of action of the force. For example, τ_x denotes a viscous stress which acts on a plan that is normal to **x** in the direction of y, which can be faces E or W in figure 4. Same notations goes with the viscous forces with an additional superscript to distinguish them from pressure forces. In the following derivation of the forces, stresses (which are the forces per unit area) are multiplied by the area of their respective plane of action in order to get and expression for the force acting on the fluid element.

The normal viscous stress force in **x** direction (f_x^v) acts in the same direction of the pressure force, however, the two forces have different origins and natures, and

therefore they are expressed in separate terms in the momentum equation. The net of f_x^v can be expressed as:

$$f_x^v = \left(\left(\tau_x + \frac{\partial \tau_x}{\partial} d\right) - \tau_x\right) d = \frac{\partial \tau_x}{\partial} d \quad (1.22)$$

and similar expressions for the normal viscous forces in **y** and **z** directions can be given as:

$$f_y^v = \left(\left(\tau_y + \frac{\partial \tau_y}{\partial} d\right) - \tau_y\right) d = \frac{\partial \tau_y}{\partial} d \quad (1.23)$$

$$f_z^v = \left(\left(\tau_z + \frac{\partial \tau_z}{\partial} d\right) - \tau_z\right) d = \frac{\partial \tau_y}{\partial} d \quad (1.24)$$

As to the shear viscous forces, there are two components in each direction. In the **x** direction, the components of shear viscous force are f_y^v and f_z^v which correspond to the shear stresses acting on faces (U & L) and (N & S), respectively. These forces can be expressed as:

$$f_y^v = \left(\left(\tau_y + \frac{\partial \tau_y}{\partial} d\right) - \tau_y\right) d = \frac{\partial \tau_y}{\partial} d \quad (1.25)$$

$$f_z^v = \left(\left(\tau_z + \frac{\partial \tau_z}{\partial} d\right) - \tau_z\right) d = \frac{\partial \tau_z}{\partial} d \quad (1.26)$$

Similarly, the shear viscous forces acting due to forces in **y** and **z** directions can be expressed, respectively, as:

$$f_x^v = \left(\left(\tau_x + \frac{\partial \tau_x}{\partial} d\right) - \tau_x\right) d = \frac{\partial \tau_x}{\partial} d \quad (1.27)$$

$$f_z^v = \left(\left(\tau_z + \frac{\partial \tau_z}{\partial} d\right) - \tau_z\right) d = \frac{\partial \tau_z}{\partial} d \quad (1.28)$$

$$f_x^v = \left(\left(\tau_x + \frac{\partial \tau_x}{\partial} d\right) - \tau_x\right) d = \frac{\partial \tau_x}{\partial} d \quad (1.29)$$

$$f_y^v = \left(\left(\tau_y + \frac{\partial \tau_y}{\partial} d\right) - \tau_y\right) d = \frac{\partial \tau_y}{\partial} d \quad (1.30)$$

Hence, by substituting equations (1.15-1.30) in $m \cdot a = \sum F$, rearranging, and eliminating the fluid element volume d from both sides, the momentum equation for incompressible fluid flow can be written in the form of:

In **x** direction: $\rho\left(\frac{\partial \boldsymbol{u}}{\partial} + (\boldsymbol{V} \cdot \nabla \boldsymbol{u})\right) = -\frac{\partial}{\partial} + \frac{\partial \tau_x}{\partial} + \frac{\partial \tau_y}{\partial} + \frac{\partial \tau_z}{\partial} + \rho \boldsymbol{f_x}$ (1.31)

In **y** direction: $\rho\left(\frac{\partial \boldsymbol{v}}{\partial} + (\boldsymbol{V} \cdot \nabla \boldsymbol{v})\right) = -\frac{\partial}{\partial} + \frac{\partial \tau_x}{\partial} + \frac{\partial \tau_y}{\partial} + \frac{\partial \tau_z}{\partial} + \rho \boldsymbol{f_y}$ (1.32)

In **z** direction: $\rho\left(\frac{\partial \boldsymbol{w}}{\partial} + (\boldsymbol{V} \cdot \nabla \boldsymbol{w})\right) = -\frac{\partial}{\partial} + \frac{\partial \tau_x}{\partial} + \frac{\partial \tau_y}{\partial} + \frac{\partial \tau_z}{\partial} + \rho \boldsymbol{f_z}$ (1.33)

By substituting the Cartesian differential operator ∇ in equations (1.31 – 1.33), the momentum equation can be written in generalized Cartesian coordinates as:

$$\rho\left(\frac{\partial \boldsymbol{V}}{\partial} + (\boldsymbol{V}\cdot\nabla\boldsymbol{V})\right) = -\nabla p + \nabla\cdot\tau_{i} + \rho\boldsymbol{f_i} \quad (1.43)$$

In the nineteenth century, Navier and Stokes derived expressions for the shear stress using Newton's assumption that shear stress is proportional to strain rate. These expressions are:

Normal stresses:

$$\tau_x = \lambda(\nabla\cdot V) + 2\mu\frac{\partial}{\partial} \quad (1.44)$$

$$\tau_y = \lambda(\nabla\cdot V) + 2\mu\frac{\partial}{\partial} \quad (1.45)$$

$$\tau_z = \lambda(\nabla\cdot V) + 2\mu\frac{\partial}{\partial} \quad (1.46)$$

where λ is the second viscosity coefficient, assumed by stokes to be equal to $-\frac{2}{3}\mu$.

Shear stresses:

$$\tau_x = \tau_y = \mu\left[\frac{\partial}{\partial} + \frac{\partial}{\partial}\right] \quad (1.47)$$

$$\tau_x = \tau_z = \mu\left[\frac{\partial}{\partial} + \frac{\partial}{\partial}\right] \quad (1.48)$$

$$\tau_y = \tau_z = \mu\left[\frac{\partial}{\partial} + \frac{\partial}{\partial}\right] \quad (1.49)$$

Substituting equations (1.44 – 1.49) in (1.43) and rearranging, we get the Navier-Stokes equations, which can be written in generalized tensor notation using Einstein's summation convention[1], as:

$$\rho\left(\frac{\partial u_i}{\partial x_i} + u_j\frac{\partial u_i}{\partial x_i}\right) = -\frac{\partial}{\partial x_i} + \mu\frac{\partial}{\partial x_i}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + S_M \quad (1.50)$$

where S_M is a source term that is used to include the smaller contributions to the viscous stress term by the second viscosity coefficient, as well as the body force. Equation (1.50) is called the Navier-Stokes equation for incompressible fluid flow.

1.3.3. Energy equation

The energy conservation principle is also applied to fluid flow in order to derive the energy equation. The derivation of the energy equation involves balance calculations such as performed for the momentum equation. However, the derivation can be found in details in relevant references and is skipped here for brevity. Thus, the energy equation for internal energy of incompressible fluid flow reads:

$$\rho\left(\frac{\partial}{\partial} + (V\cdot\nabla\boldsymbol{I}\right) = -p\nabla\cdot V + k(\nabla\cdot\nabla\mathrm{T}) + \Phi + S_I \quad (1.51)$$

[1] Einstein's summation convention states that the differential Cartesian operator ∇ can be written as $\frac{\partial}{\partial x_i}$ with $i = 1,2,3$ representing the dimensions **x**, **y** and **z**.

where the left hand side is the rate of change of energy represented by the local and convective derivatives of internal energy ($I = h - \frac{p}{\rho}$), the first term on the right hand side represents the rate of work done on the fluid element by the pressure forces, the second term on RHS is the net rate of heat transfer to the fluid element, the third term (Φ) is the viscous dissipation term, and S_I is a source term.

The finite Volume Method

The governing equations of fluid flow are highly nonlinear, partial differential equations of field functions. Therefore, they only have exact solutions in very simple and special cases that are seldom to find in practical applications. Consequently, in order to solve these equations for realistic flows, computers must be involved. In order to make these equations suitable to be solved by computers, they have to be linearized. This is because binary computers can only perform logical operations, such as addition, subtraction, multiplication, and so on. The linearization of the governing equations usually involves complex operations and processes. The outcome is a system of linear equations that can be solved given a solution domain space (the fluid flow volume, which must be linearized as well), proper boundary conditions, and a solution algorithm. The solution approach in such case is of course an iterative one, where assumptions of the solution are made and then corrected in sequential steps by submitting to the solution algorithm, boundary conditions and some other sets of rules. Figure 5 schematically explains the components of any computational fluid dynamics tool.

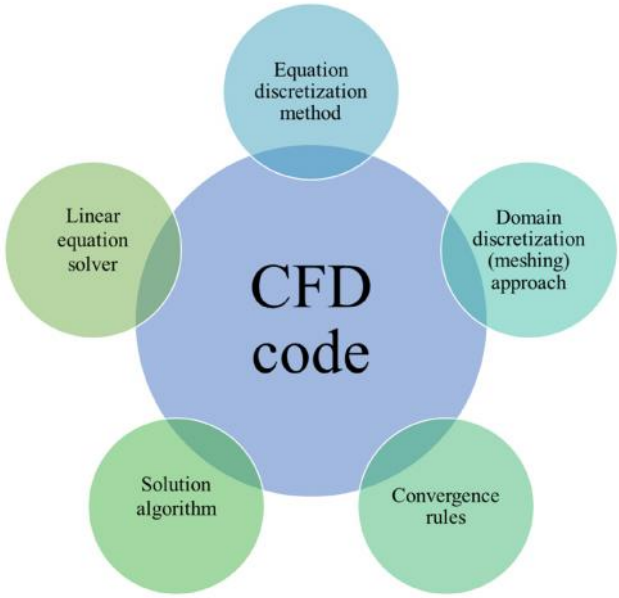


Figure 5. Schematic diagram for the components of a CFD code

Equation discretization methods are used to linearize the partial differential equations into sets (i.e. matrices) of linear equations. In order to accomplish this, the solution domain must be expressed in terms of small finite intervals. This is achieved by a domain discretization approach, which defines the nature and shape of these intervals, and also the locations at which different flow field variables will be computed. The linear equation solver, then, applies the adopted solution algorithm in order to solve the resulting matrices given proper boundary and initial conditions. The solvers proceeds until the convergence rules are achieved. The solution algorithm defines the sequence of solving different governing equations, while the linear equation solver actually solves them in matrices form.

Today, there are numerous general purpose CFD codes, each of which contains different collection of selections to set up each part of the code for solving specific problems.

Commercial codes such as ANSYS Fluent® provide such selections in a user-friendly interface and a smooth workflow, however with minimum access to the code itself. Community based open source toolboxes such as OpenFOAM®, nevertheless, provide maximum access to the code without a user friendly interface. However, it is valid for commercial and open source CFD codes that the Finite Volume Method (FVM), with its subset tools, schemes, and algorithms has become the most used method in CFD today. Some experts would say that 85% of the CFD simulations conducted today are conducted using FVM. For that reason, we have decided to focus only on such method for the scope of these notes.

The Finite Volume Method (FVM) is one of the most versatile discretization techniques used in CFD. Based on the control volume formulation of analytical fluid dynamics, the first step in the FVM is to divide the domain into a number of control volumes (aka cells) where the variable of interest is located at the centroid of the control volume. The divided domain is usually called “grid” or “computational domain” or simply “the mesh”, see figure 6.The next step is to integrate the differential form of the governing equations over each control volume. Theoretically, this is accomplished with the help of Gauss and Stokes theorems. Interpolation schemes are then assumed in order to describe the variation of the concerned variable between cell centroids. The resulting equation is called the discretized or discretization equation. This equation is basically an algebraic form of the governing/transport equation. In this manner, the discretization equation expresses the conservation principle for the variable (velocity, pressure, temperature…etc.) inside the control volume.

The most compelling feature of the FVM is that the resulting solution satisfies the conservation of quantities such as mass, momentum, energy, and species. This is exactly satisfied for any control volume as well as for the whole computational domain and for any number of control volumes. Even coarse grid solution exhibits exact integral balances, it is flexible in terms of both geometry and variety of physical phenomena and it is directly relatable to physical quantities.

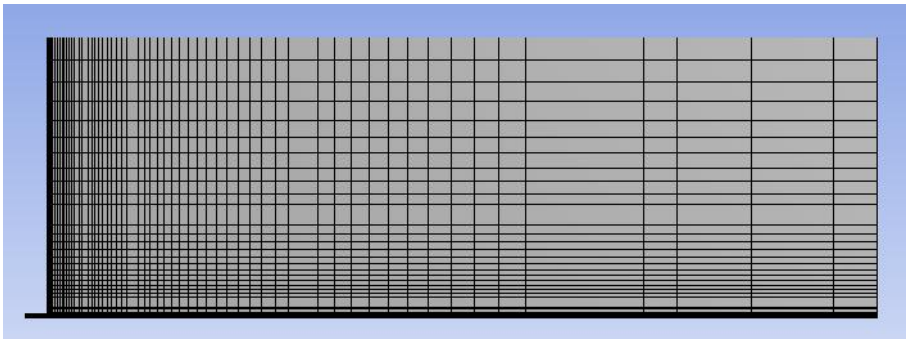


Figure 6. Computational domain for submerged jet flow

1.4. Gauss (divergence) theorem

Gauss theorem states that the integral on a volume V is equal to the integral on the surface S which bounds V. If n is unit normal vector on S, assumed to be positive outward the area enclosed by S, Stokes theorem can be expressed as:

$$\int_V \frac{\partial}{\partial x_i} d = \int_S \phi n_i d \qquad (2.1)$$

where *i*=1,2,3, ϕ is a field scalar function or a component of a vector or tensor fields and $n_i = \cos(n \cdot x_i)$.

1.5. Stokes theorem

Stokes theorem states that the integral of an open surface ***S*** is equal to the integral of a curve ***C*** which bounds this surface. As shown in figure 7, and if $\vec{V} = iu + jv + kw$ is a velocity vector in the Cartesian coordinates, Stokes theorem can be expressed as:

$$\int_S \left[\left(\frac{\partial}{\partial} - \frac{\partial}{\partial}\right)n_1 + \left(\frac{\partial}{\partial} - \frac{\partial}{\partial}\right)n_2 + \left(\frac{\partial}{\partial} - \frac{\partial}{\partial}\right)n_3\right] d \quad = \oint_C (u \quad + v \quad + w \quad) \quad (2.2)$$

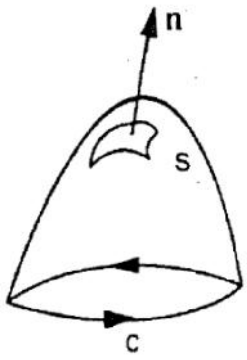


Figure 7. Open surface S with a bounding surface C

1.6. The finite volume method

Consider the governing equation for an arbitrary flow variable ϕ, in the form:

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot (\rho \mathbf{u}\phi) = \nabla \cdot (\Gamma \nabla \phi) + S_\phi \quad (2.3)$$

When the unsteady and convection terms are neglected, equation (2.3) becomes:

$$\nabla \cdot (\Gamma \nabla \phi) + S_\phi = 0 \quad (2.4)$$

where Γ is the diffusion coefficient and S_ϕ is a source term of the flow field variable.

If equation (2.4) is integrated over a control volume, by applying Stokes theorem, the control volume integration can be expressed as:

$$\int_{CV} \nabla \cdot (\Gamma \nabla \phi) dV + \int_{CV} S_\phi dV = \int_A \mathbf{n} \cdot (\Gamma \nabla \phi) dA + \int_{CV} S_\phi dV = 0 \quad (2.5)$$

where *V* is the control volume and *A* is the area bounding the control volume. Consider the steady state diffusion of a property φ in a one-dimensional domain:

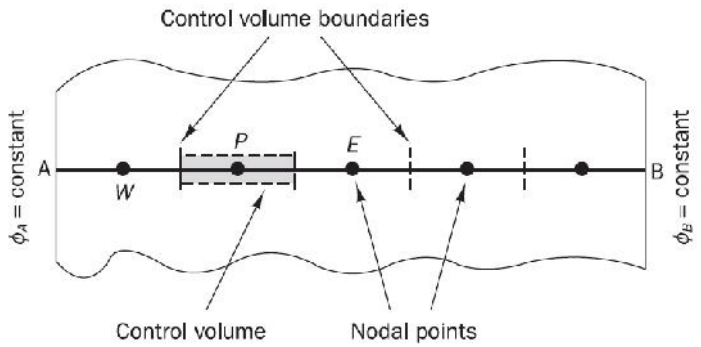


Figure 8 Control volumes for a one dimensional, steady-state diffusion problem

- The process is governed by $\frac{d}{dx}\left(\Gamma\frac{d¤}{dx}\right)+S=0$

1.6.1. Grid generation

- ✓ The first step in the finite volume method is to divide the domain into discrete control volumes.
- ✓ Each node is surrounded by a control volume or cell.
  - → A general nodal point is identified by P.
  - → Its neighbors in a one-dimensional geometry, the nodes to the west and east, are identified by W and E respectively.
  - → The west side face of the control volume is referred to by "w" and th east side control volume face by "e".
  - → The distances between the nodes W and P and between nodes P and E are identified by δx_{WP} and δx_{PE} respectively.
  - → The control volume width is $\Delta x = \delta x_{we}$.

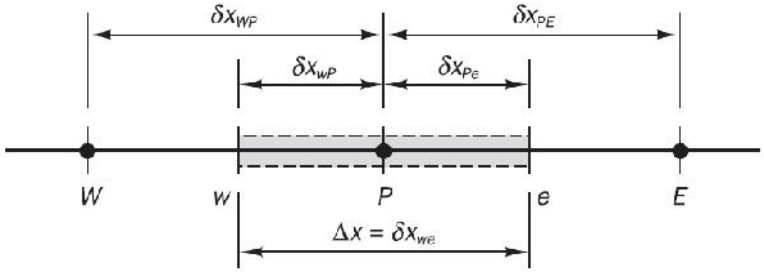


Figure 9. Computational domain specification for the 1-D steady diffusion problem

1.6.2. Discretization

- ✓ The key step of the finite volume method is the integration of the governing equation over a control volume to yield a discretized equation at its nodal point P.
- ✓ For the control volume defined:

"The diffusive flux of φ leaving the east face minus the diffusive flux of φ entering the west face is equal to the generation of φ"

$$\int_{\Delta V}\frac{d}{dx}\left(\Gamma\frac{d¤}{dx}\right)dV+\int_{\Delta V}S_¤\,dV=\left(\Gamma A\frac{d¤}{dx}\right)_e-\left(\Gamma A\frac{d¤}{dx}\right)_w+\bar{S}\Delta V=0 \qquad (2.6)$$

where A is the cross-sectional area of the control volume face, ΔV is the volume and $\bar{S}$ is the average value of source S over the control volume.

→ To calculate gradient (and hence fluxes) at the control volume faces, an approximate distribution of properties between nodal points is used.

→ Linear approximation is the simplest way of calculating interface values and the gradients and is called central differencing:

$$\Gamma_w = \frac{\Gamma_W + \Gamma_P}{2} \text{ and } \Gamma_e = \frac{\Gamma_P + \Gamma_E}{2}$$

→ The diffusive flux terms are:

$$\left(\Gamma A \frac{d¤}{dx}\right)_w = \Gamma_w A_w \left(\frac{¤_P - ¤_W}{¢x_{WP}}\right) \text{ and } \left(\Gamma A \frac{d¤}{dx}\right)_e = \Gamma_e A_e \left(\frac{¤_E - ¤_P}{¢x_{PE}}\right)$$

(2.7)

→ The source term by means of a linear form:

$$\bar{S}\Delta V = S_u + S_p ¤_P$$

(2.8)

1.6.3. Preparation of the linear equations system

The equation is then:

$$\left(\Gamma A \frac{d¤}{dx}\right)_e - \left(\Gamma A \frac{d¤}{dx}\right)_w + \bar{S}\Delta V = \Gamma_e A_e \left(\frac{¤_E - ¤_P}{¢x_{PE}}\right) - \Gamma_w A_w \left(\frac{¤_P - ¤_W}{¢x_{WP}}\right) + S_u + S_p ¤_P = 0$$

After rearranging:

$$\left(\frac{\Gamma_e}{¢x_{PE}} A_e + \frac{\Gamma_w}{¢x_{WP}} A_w - S_p\right) ¤_P = \left(\frac{\Gamma_w}{¢x_{WP}} A_w\right) ¤_W + \left(\frac{\Gamma_e}{¢x_{PE}} A_e\right) ¤_E + S_u \qquad (2.9)$$

or:

$$a_P ¤_P = a_W ¤_W + a_E ¤_E + S_u \qquad (2.10)$$

with

$$a_W = \frac{\Gamma_w}{¢x_{WP}} A_w \qquad a_E = \frac{\Gamma_e}{¢x_{PE}} A_e \qquad a_P = a_W + a_E - S_P \qquad (2.11)$$

1.6.4. Application of boundary conditions

The discretized equations must be set up at each of the nodal points in order to solve a problem.

→ For control volumes that are adjacent to the domain boundaries, the general discretized equation is modified to incorporate boundary conditions.

→ The resulting system of linear algebraic equations is solved to obtain the distribution of the property φ at nodal points.

Example I – One dimensional heat conduction in a rod

The equation governing one-dimensional steady state conductive heat transfer is:

$$\frac{d}{dx}\left(k \frac{dT}{dx}\right) + S = 0$$

Consider the problem of source-free heat conduction in an insulated rod whose ends are maintained at constant temperatures of 100˚C and 500˚C respectively. The one-dimensional problem is governed by:

$$\frac{d}{dx}\left(k\frac{dT}{dx}\right)=0$$

Calculate the steady state temperature distribution in the rod. Thermal conductivity *k* = 1000 W/mK, cross-sectional area *A* is 10 x 10^{-3} m^2.

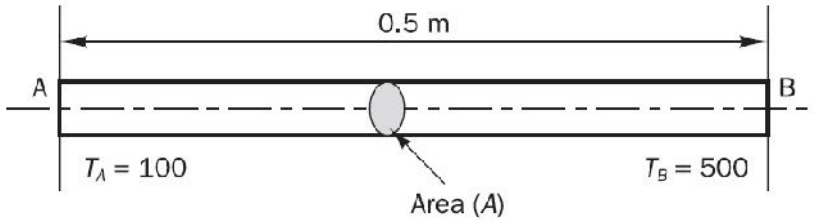


Figure 10. Schematic of the 1-D steady heat conduction in a rod

1) Divide the length of the rod into 5 equal control volumes. This gives δx = 0.1 m.
2) Discretised equations can be readily written for control volumes surrounding these nodes:

$$\left(\frac{k_e}{¢x_{PE}}A_e+\frac{k_w}{¢x_{WP}}A_w\right)T_P=\left(\frac{k_w}{¢x_{WP}}A_w\right)T_W+\left(\frac{k_e}{¢x_{PE}}A_e\right)T_E$$

3) For nodes 2, 3, and 4

   Compared with the general form:

   $a_P T_P = a_W T_W + a_E T_E$

   we have:

   $k_e = k_w = k \quad A_e = A_w = A$

   $a_W = \frac{k}{¢x}A \qquad a_E = \frac{k}{¢x}A \qquad a_P = a_W + a_E$

4) For the boundary nodes 1 and 5, from:

$$\left(\frac{k_e}{¢x_{PE}}A_e+\frac{k_w}{¢x_{WP}}A_w\right)T_P=\left(\frac{k_w}{¢x_{WP}}A_w\right)T_W+\left(\frac{k_e}{¢x_{PE}}A_e\right)T_E$$

- For node 1, we have:

$$\left(\frac{k}{¢x}A+\frac{k}{\frac{1}{2}¢x}A\right)T_P=\left(\frac{k}{\frac{1}{2}¢x}A\right)T_A+\left(\frac{k}{¢x}A\right)T_E$$

or

$$kA\left(\frac{T_E-T_P}{¢x}\right)-kA\left(\frac{T_P-T_A}{\frac{1}{2}¢x}\right)=0$$

or

$$\left(\frac{k}{¢x}A+\frac{2k}{¢x}A\right)T_P=0T_W+\left(\frac{k}{¢x}A\right)T_E+\left(\frac{2k}{¢x}A\right)T_A$$

- For node 5, we have:

$$kA\left(\frac{T_B - T_P}{\frac{1}{2}\Delta x}\right) - kA\left(\frac{T_P - T_W}{\Delta x}\right) = 0$$

or

$$\left(\frac{k}{\Delta x}A + \frac{2k}{\Delta x}A\right)T_P = \left(\frac{k}{\Delta x}A\right)T_W + 0T_E + \left(\frac{2k}{\Delta x}A\right)T_B$$

- Compared with the general form:

  $a_P T_P = a_W T_W + a_E T_E + S_u$

  we have

  $a_E = 0 \qquad a_W = \frac{k}{\Delta x}A \qquad a_P = a_W + a_E - S_P \qquad S_P = -\frac{2k}{\Delta x}A$

  $S_u = \frac{2k}{\Delta x}AT_B$

5) The values for each discretized equation are:

Node	a_W	a_E	S_u	S_P	$a_P = a_W + a_E - S_P$
1	0	100	$200T_A$	−200	300
2	100	100	0	0	200
3	100	100	0	0	200
4	100	100	0	0	200
5	100	0	$200T_B$	−200	300

6) The resulting set of algebraic equations:

$300T_1 = 100T_2 + 200T_A$

$200T_2 = 100T_1 + 100T_3$

$200T_3 = 100T_2 + 100T_4$

$200T_4 = 100T_3 + 100T_5$

$300T_5 = 100T_4 + 200T_B$

or

$$\begin{bmatrix} 300 & -100 & 0 & 0 & 0 \\ -100 & 200 & -100 & 0 & 0 \\ 0 & -100 & 200 & -100 & 0 \\ 0 & 0 & -100 & 200 & -100 \\ 0 & 0 & 0 & -100 & 300 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} 200T_A \\ 0 \\ 0 \\ 0 \\ 200T_B \end{bmatrix}$$

7) For $T_A = 100$ and $T_B = 500$

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} 140 \\ 220 \\ 200 \\ 380 \\ 460 \end{bmatrix}$$

Elements of a CFD solver

1.7. Computational grid

The discrete locations at which the variables are to be calculated are defined by the numerical grid which is essentially a discrete representation of the geometric domain on which the problem is to be solved. It divides the solution domain into a finite number of subdomains (elements, control volumes etc.). Some of the options available are the following:

1.7.1. Structured grids

Regular or structured grids consist of families of grid lines with the property that members of a single family do not cross each other and cross each member of the other families only once. This allows the lines of a given set to be numbered consecutively. The position of any grid point (or control volume) within the domain is uniquely identified by a set of two (in 2D) or three (in 3D) indices, e.g. (i, j, k). This is the simplest grid structure, since it is logically equivalent to a Cartesian grid. Each point has four nearest neighbors in two dimensions and six in three dimensions; one of the indices of each neighbor of point P (indices i, j, k) differs by f 1 from the corresponding index of P. An example of a structured 2D grid is shown in Figure 11. This neighbor connectivity simplifies programming and the matrix of the algebraic equation system has a regular structure, which can be exploited in developing a solution technique.

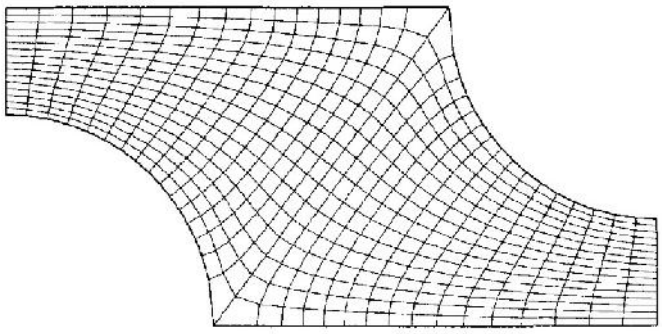


Figure 11. Example of a 2D, structured, non-orthogonal grid, designed for calculation of flow in a symmetry segment of a staggered tube bank

1.7.2. Unstructured grids

For very complex geometries, the most flexible type of grid is one which can fit an arbitrary solution domain boundary. In principle, such grids could be used with any discretization scheme, but they are best adapted to the finite volume and finite element approaches. The elements or control volumes may have any shape; nor is there a restriction on the number of neighbor elements or nodes. In practice, grids made of triangles or quadrilaterals in 2D, and tetrahedral or hexahedra in 3D are most often used. Such grids can be generated automatically by existing algorithms. If desired, the grid can be made orthogonal, the aspect ratio is easily controlled, and the grid may be easily locally refined. The advantage of flexibility is offset by the disadvantage of the

irregularity of the data structure. Node locations and neighbor connections need be specified explicitly. The matrix of the algebraic equation system no longer has regular, diagonal structure; the band width needs to be reduced by reordering of the points. The solvers for the algebraic equation systems are usually slower than those for regular grids.

Unstructured grids are usually used with finite element methods and, increasingly, with finite volume methods. Computer codes for unstructured grids are more flexible. They need not be changed when the grid is locally refined, or when elements or control volumes of different shapes are used. However, grid generation and pre-processing are usually much more difficult. The finite volume method presented in this book is applicable to unstructured grids. An example of an unstructured grid is shown in Figure 12.

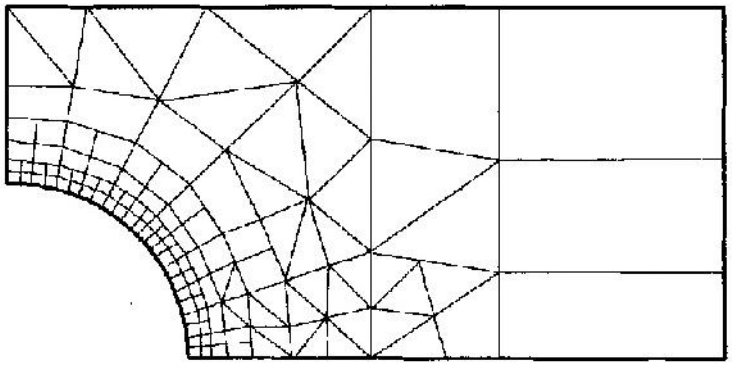


Figure 12. Example of a 2D unstructured grid

1.8. Numerical discretization schemes

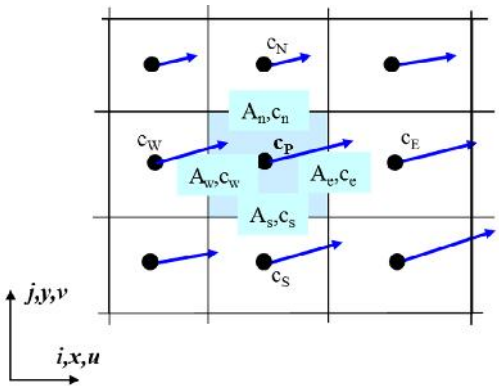
Numerical schemes are basically methods used to interpolate flow field variables across the computational domain. The need for such schemes arise because the system of equations resulting from applying the FVM on the computational grid poses a boundary-value problem. In such problem, the values of the flow field variables are known on the boundary cells of the grid, and unknown in other cells. The coefficients of different terms of the system of equations are references to the computational grid indices, hence, the only unknown is the flow field variable. Let us consider the following example of a 2-D steady diffusion-convection problem.

Let the species transport equation for incompressible steady flow be expressed as:

$$\nabla \cdot (u_i c) = \nabla \cdot (\mathrm{D}\nabla \mathrm{c}) + S \qquad (3.1)$$

where c is the species concentration, D is the diffusion coefficient and S is a source term.

Consider the flow field given by the computational domain schematically shown in figure 13. In such case, the differential operator becomes $\nabla = \frac{\partial}{\partial} + \frac{\partial}{\partial}$



Notation

A_w, A_n, A_e, A_s : areas of the faces

c_w, c_n, c_e, c_s : concentrations at the faces

c_W, c_N, c_E, c_S : concentrations at the cell centers

$u_w, u_n, u_e, u_s, v_w, v_n, v_e, v_s$: velocities at the faces

$u_W, u_N, u_E, u_S, v_W, v_N, v_E, v_S$: velocities at the cell centers

S_P: source in cell P

D: diffusion coefficient

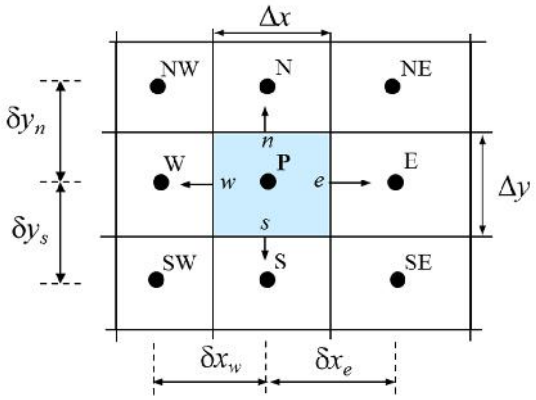


Figure 13. Schematic of the computational domain of a steady 2-D diffusion convection problem. The upper figure shows the cell indices and the lower shows the cell size and neighbor cells indices.

The integration over the control volume is given by:

Integration of the first derivative:

$$(A_e u_e c_e - A_w u_w c_w) + (A_n u_n c_n - A_s u_s c_s) = \left(DA_e \frac{d}{d}_e - DA_w \frac{d}{d}_w\right) + \left(DA_n \frac{d}{d}_n - DA_s \frac{d}{d}_s\right) + S_p \quad (3.2)$$

Integration of the second derivative:

$$(A_e u_e c_e - A_w u_w c_w) + (A_n u_n c_n - A_s u_s c_s) = \left(DA_e \left(\frac{c_e - c_p}{\delta x_e}\right) - DA_w \left(\frac{c_w - c_p}{\delta x_w}\right)\right) + \left(DA_n \left(\frac{c_n - c_p}{\delta y_n}\right) - DA_s \left(\frac{c_s - c_p}{\delta y_s}\right)\right) + S_p \quad (3.3)$$

By rearranging equation (3.3), we get:

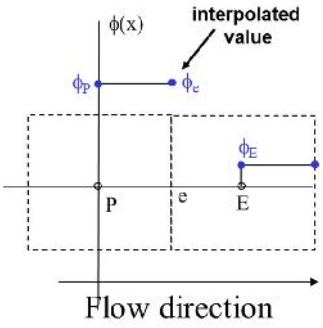
$$\begin{aligned} c_P(A_n v_P + A_e u_P + \mathbf{D}A_w / \¢\, x_w + \mathbf{D}A_n / \¢\, y_p + \mathbf{D}A_e / \¢\, x_e + \mathbf{D}A_s / \¢\, y_s) \;=\; & c_W(A_w u_W + \mathbf{D}A_w / \¢\, x_w) + \\ & c_N(\mathbf{D}A_n / \¢\, y_n) + \\ & c_E(\mathbf{D}A_e / \¢\, x_e) + \\ & c_S(A_s v_S + \mathbf{D}A_s / \¢\, y_s) + \\ & S_P \end{aligned} \quad (3.4)$$

which can be simplified, by the use of subscript ***nb*** to indicate any neighboring cell, to become:

$$a_P c_P = a_W c_W + a_N c_N + a_E c_E + a_S c_S + b$$
$$= \sum_{nb} a_{nb} c_{nb} + b \qquad (3.5)$$

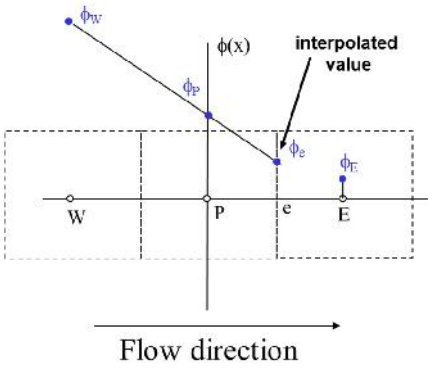
The coefficients a_{nb} and b will be different for every cell in the domain at every iteration. The species concentration field can be calculated by recalculating c_P from this equation iteratively for all cells in the domain. It is clear that to solve equation (3.5), there must be relations between the cell centers and call face values of the flow field variables. The most widely used *interpolation* schemes can be summarized as following:

1.8.1. First order upwind scheme



This is the simplest numerical scheme. It is the method that we used earlier in the discretization example. We assume that the value of ϕ at the face is the same as the cell center value in the cell upstream of the face. The main advantages are that it is easy to implement and that it results in very stable calculations, but it also very diffusive. Gradients in the flow field tend to be smeared out, as we will show later. This is often the best scheme to start calculations with.

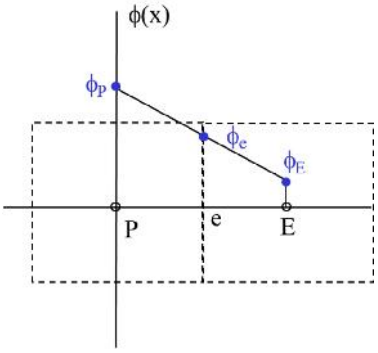
1.8.2. Second order upwind scheme



We determine the value of ¤ from the cell values in the two cells upstream of the face. This is more accurate than the first order upwind scheme, but in regions with strong gradients it can result in face values that are outside of the range of cell values. It is then necessary to apply limiters to the predicted face values. There are many different ways to implement this, but second-order upwind with limiters is one of the more popular numerical schemes because of its combination of accuracy and stability.

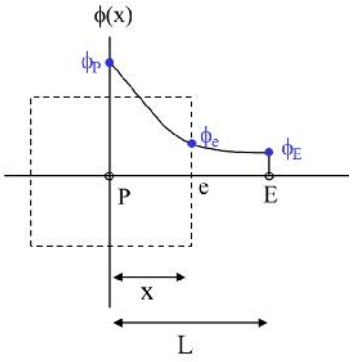
1.8.3. Central differencing scheme

The value of ϕ at the face is determined by linear interpolation between the cell center values. This is more accurate than the first order upwind scheme, but it leads to oscillations in the solution or divergence if the local Peclet number is larger than 2. The Peclet number is the ratio between convective and diffusive transport: $P = \frac{\rho}{D}$. It is common to then switch to first order upwind in cells where Pe>2. Such an approach is called a hybrid scheme.



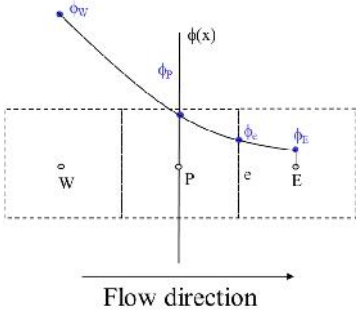
1.8.4. Power law scheme

This is based on the analytical solution of the one-dimensional convection-diffusion equation. The face value is determined from an exponential profile through the cell values. The exponential profile is approximated by the following power law equation: $\phi_e = \phi_P - \frac{(1-0.1Pe)^5}{P}(\phi_E - \phi_P)$ where Pe is the local Peclet number.



1.8.5. QUICK Scheme

QUICK stands for Quadratic Upwind Interpolation for Convective Kinetics. A quadratic curve is fitted through two upstream nodes and one downstream node. This is a very accurate scheme, but in regions with strong gradients, overshoots and undershoots can occur. This can lead to stability problems in the calculation.



1.8.6. Accuracy of discretization schemes

Each of the previously discussed numerical schemes assumes some shape of the function ¤. These functions can be approximated by Taylor series polynomials:

$$¤(x_e) = ¤(x_P) + \frac{¤'(x_P)}{1!}(x_e - x_P) + \frac{¤''(x_P)}{2!}(x_e - x_P)^2 + ... + \frac{¤^n(x_P)}{n!}(x_e - x_P)^n + ... \quad (3.6)$$

The first order upwind scheme only uses the constant and ignores the first derivative and consecutive terms. This scheme is therefore considered first order accurate. For high Peclet numbers the power law scheme reduces to the first order upwind scheme, so it is also considered first order accurate. The central differencing scheme and second order upwind scheme do include the first order derivative, but ignore the second order

derivative. These schemes are therefore considered second order accurate. QUICK does take the second order derivative into account, but ignores the third order derivative. This is then considered third order accurate.

1.9. Linear equation solution methods

Discretization yields a large system of non-linear algebraic equations. The method of solution depends on the problem. For unsteady flows, methods based on those used for initial value problems for ordinary differential equations (marching in time) are used. At each time step an elliptic problem has to be solved. Steady flow problems are usually solved by pseudo-time marching or an equivalent iteration scheme. Since the equations are non-linear, an iteration scheme is used to solve them. These methods use successive linearization of the equations and the resulting linear systems are almost always solved by iterative techniques. The choice of solver depends on the grid type and the number of nodes involved in each algebraic equation.

It was shown how the convection-diffusion equation may be discretized using FD and FV methods. In either case, the result of the discretization process is a system of algebraic equations, which are linear or non-linear according to the nature of the partial differential equation(s) from which they are derived. In the non-linear case, the discretized equations must be solved by an iterative technique that involves guessing a solution, linearizing the equations about that solution, and improving the solution; the process is repeated until a converged result is obtained. So, whether the equations are linear or not, efficient methods for solving linear systems of algebraic equations are needed. The matrices derived from partial differential equations are always sparse, i.e. most of their elements are zero. Some methods for solving the equations that arise when structured grids are used will be described below; all of the non-zero elements of the matrices then lie on a small number of well-defined diagonals; we may take advantage of this structure. Some of the methods are applicable to matrices arising from unstructured grids as well. The structure of the coefficient matrix for a 2D problem discretized with a five point approximation (upwind or central difference) is shown in Fig. 3.5. The algebraic equation for one CV or grid node is given by Equation (1.18), and the matrix version of the complete problem is given by Equation (1.19)

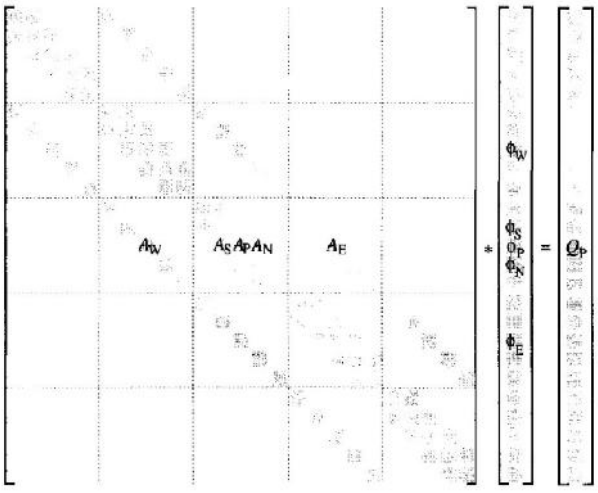


Figure 14. Structure of the matrix for a five-point computational molecule (nonzero entries in the coefficient matrix on five diagonals are shaded; each horizontal set of boxes corresponds to one grid line).

1.9.1. Direct Methods

The matrix A is assumed to be very sparse. In fact, the most complicated matrix we shall encounter is a banded matrix of block type; this greatly simplifies the task of solution but we shall briefly review methods for general matrices as methods for sparse matrices are closely related to them. For the description of methods designed to deal with full matrices, use of full-matrix is more sensible and will be adopted.

1.9.1.1. Gauss Elimination

The basic method for solving linear systems of algebraic equations is Gauss elimination. Its basis is the systematic reduction of large systems of equations to smaller ones. In this procedure, the elements of the matrix are modified but, as the dependent variable names do not change, it is convenient to describe the method in terms of the matrix alone:

$$A = \begin{pmatrix} A_1 & A_1 & A_1 & \cdots & A_{1n} \\ A_2 & A_2 & A_2 & \cdots & A_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \cdots & A_n \end{pmatrix} \qquad 3.7$$

The heart of the algorithm is the technique for eliminating A_2 i.e., replacing it with a zero. This is accomplished by multiplying the first equation (first row of the matrix) by $A_2 \ / A_1$ and subtracting it from the second row or equation; in the process, all of the elements in the second row of the matrix are modified as is the second element of the forcing vector on the right hand side of the equation. The other elements of the first column of the matrix, A_3 , A_4 . . . , A_{n1} are treated similarly; for example, to eliminate Ail, the first row of the matrix is multiplied by A_{i1}/A_1 and subtracted from the i[th] row. By systematically proceeding down the first column of the matrix, all of the elements below A_1 are eliminated. When this process is complete, none of the equations 2, 3, ... , n contain the variable ϕ_1; they are a set of n – 1 equations for the variables ϕ_2, ϕ_3, . . . , ϕ_n. The same procedure is then applied to this smaller set of equations - all of the elements below A_2 in the second column are eliminated.

This procedure is carried out for columns 1,2,3, . . . , n - 1. After this process is complete, the original matrix has been replaced by an upper triangular one:

$$U = \begin{pmatrix} A_1 & A_1 & A_1 & \cdots & A_{1n} \\ 0 & A_2 & A_2 & \cdots & A_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \qquad 3.8$$

All of the elements except those in the first row differ from those in the original matrix A. As the elements of the original matrix will never be needed again, it is efficient to store the modified elements in place of the original ones. (In the rare case requiring that the original matrix be saved, a copy can be created prior to starting the elimination procedure.) This portion of the algorithm just described is called forward elimination. The elements on the right hand side of the equation, Q_i, are also modified in this procedure. The upper triangular system of equations resulting from forward elimination is easily solved. The last equation contains only one variable, ϕ_n, and is readily solved:

$$\phi_n = \frac{Q_n}{A_n} \qquad 3.9$$

The next to last equation contains only ϕ_{n-1} and ϕ_n, and, once ϕ_n, is known, it can be solved for ϕ_{n-1} Proceeding upward in this manner, each equation is solved in turn; the ith equation yields ϕ_i:

$$\phi_i = \frac{Q_i - \sum_{k=i+1}^{n} A_{ij}\,\phi_k}{A_n} \qquad 3.10$$

The right hand side is calculable because all of the ϕ_k appearing in the sum have already been evaluated. In this way, all of the variables may be computed. The part of the Gauss elimination algorithm which starts with the triangular matrix and computes the unknowns is called back substitution. It is not difficult to show that, for large n, the number of operations required to solve a linear system of n equations by Gauss elimination is proportional to $n^3/3$. The bulk of this effort is in the forward elimination phase; the back substitution requires only $n^2/2$ arithmetic operations and is much less costly than the forward elimination. Gauss elimination is thus expensive but, for full matrices, it is as good as any method available. The high cost of Gauss elimination provides incentive to search for more efficient special solvers for matrices such as the sparse ones arising from the discretization of differential equations. For large systems that are not sparse, Gauss elimination is susceptible to accumulation of errors which makes it unreliable if not modified. The addition of pivoting or interchange of rows in order to make the pivot elements (the diagonal elements that appear in the denominators) as large as possible, keeps the error growth in check. Fortunately, for sparse matrices, error accumulation is rarely a problem so this issue is not important here. Gauss elimination does not vectorize or parallelize well and is rarely used without modification in CFD problems.

1.9.1.2. LU Decomposition

A number of variations on Gauss elimination have been proposed. Most are of little interest here. One variant of value to CFD is LU decomposition. It is presented without derivation. We have seen that, in Gauss elimination, forward elimination reduces a full matrix to an upper triangular one. This process can be carried out in a more formal manner by multiplying the original matrix A by a lower triangular matrix. By itself, this is of little interest but, as the inverse of a lower triangular matrix is also lower triangular, this result shows that any matrix A, subject to some limitations that can be ignored here, can be factored into the product of lower (L) and upper (U) triangular matrices:

$$A = L \qquad 3.11$$

To make the factorization unique, we require that the diagonal elements of L, L_{ii}, all be unity; alternatively, one could require the diagonal elements of U to be unity. What makes this factorization useful is that it is easily constructed. The upper triangular matrix U is precisely the one produced by the forward phase of Gauss elimination. Furthermore, the elements of L are the multiplicative factors (e.g. A_{ji}/A_{ii}) used in the elimination process. This allows the factorization to be constructed by a minor modification of Gauss elimination. Furthermore, the elements of L and *U* can be stored where the elements of A were. The existence of this factorization allows the solution of the system of equations $A = Q$ in two stages. With the definition:

$$U = Y \qquad 3.12$$

Thus, the system of equations becomes:

$$L \quad = Q \qquad 3.13$$

The latter set of equations can be solved by a variation of the method used in the backward substitution phase of Gauss elimination in which one starts from the top rather than the bottom of the system. Once Equation (3.13) has been solved for Y, Equation (3.12), which is identical to the triangular system solved in the back substitution phase of Gauss elimination, can be solved for ϕ. The advantage of LU factorization over Gauss elimination is that the factorization can be performed without knowing the vector Q. As a result, if many systems involving the same matrix are to be solved, considerable savings can be obtained by performing the factorization first; the systems can then be solved as required. As we shall see below, variations on LU factorization are the basis of some of the better iterative methods of solving systems of linear equations; this is the principal reason for introducing it here.

1.10. Iterative methods

1.10.1. Basic Concept

Any system of equations can be solved by Gauss elimination or LU decomposition. Unfortunately, the triangular factors of sparse matrices are not sparse, so the cost of these methods is quite high. Furthermore, the discretization error is usually much larger than the accuracy of the computer arithmetic so there is no reason to solve the system that accurately. Solution to somewhat more accuracy than that of the discretization scheme suffices. This leaves an opening for iterative methods. They are used out of necessity for non-linear problems, but they are just as valuable for sparse linear systems. In an iterative method, one guesses a solution, and uses the equation to systematically improve it. If each iteration is cheap and the number of iterations is small, an iterative solver may cost less than a direct method. In CFD problems this is usually the case. Consider the matrix problem represented by Equation ($A \quad = Q$) which might result from FV approximation of a flow problem. After n iterations we have an approximate solution ϕ^n which does not satisfy these equations exactly. Instead, there is a non-zero residual ρ^n:

$$A\phi^n = Q - \rho^n \qquad 3.14$$

By subtracting this equation from Equation ($A \quad = Q$), we obtain a relation between the iteration error defined by:

$$\varepsilon^n = \phi - \phi^n \qquad 3.15$$

where ϕ is the converged solution, and the residual:

$$A\varepsilon^n = \rho^n \qquad 3.16$$

The purpose of the iteration procedure is to drive the residual to zero; in the process, ε also becomes zero. To see how this can be done, consider an iterative scheme for a linear system; such a scheme can be written:

$$M\phi^{n+1} = N\phi^n + B \qquad 3.17$$

An alternative version of this iterative method may be obtained by subtracting $M\phi^n$ from each side of Equation (3.17). We obtain:

$$M(\phi^{n+1} - \phi^n) = B - (M - N)\phi^n \quad \text{or} \quad M\delta^n = \rho^n \qquad 3.18$$

where $\delta^n = \phi^{n+1} - \phi^n$ is called the correction or update and is an approximation to the iteration error. For an iterative method to be effective, solving the system (3.18) must be cheap and the method must converge rapidly. Inexpensive iteration requires that computation of $N \quad ^n$

and solution of the system must both be easy to perform. The first requirement is easily met; since A is sparse, N is also sparse, and computation of N^{n} is simple. The second requirement means that the iteration matrix M must be easily inverted; from a practical point of view, M should be diagonal, tridiagonal, triangular, or, perhaps, block tridiagonal or triangular; another possibility is described below. For rapid convergence, M should be a good approximation to A, making N4 small in some sense.

1.10.2. The Gauss-Seidel point-by-point method

The simplest of all iterative methods is the Gauss-Seidel method in which the values of the variable are calculated by visiting each grid point in a certain order. Only one set of T's is held in computer storage. In the beginning, these represent the initial guess or values from the previous iteration. As each grid point is visited, the corresponding value of T in the computer storage is altered as follows: If the discretization equation is written as

$$a_p T_p = \sum a_n \; T_n \; + b \qquad 3.19$$

where the subscript n denotes a neighbor point, then T_p at the visited grid point is calculated from

$$T_p = \frac{\sum a_n \; T_n^* \; + b}{a_p} \qquad 3.20$$

where T_n^* stands for the neighbor-point value present in the computer storage. For neighbors that have already been visited during the current iteration, T_n^* is the freshly calculated value; for yet-to-be-visited neighbors, T_n^* is the value from the previous iteration. In any case, T_n^* is the latest available value for the neighbor-point temperature. When all grid points have been visited in this manner, one iteration of the Gauss-Seidel method is complete.

We thus conclude that the Gauss-Seidel method does not always converge. Indeed, a criterion has been formulated by Scarborough (1958) that, when satisfied, guarantees the convergence of the Gauss-Seidel method. We shall state it without proof and discus~ its implications. The Scarborough criterion. A sufficient condition for the convergence of the Gauss-Seidel method is:

$$\frac{\sum |a_n|}{|a_p|} \begin{cases} \leq 1 & f \quad a \quad e \\ < 1 & f \quad a \quad l\epsilon \quad o \quad e \end{cases} \qquad 3.21$$

1.10.3. Incomplete LU Decomposition: Stone's Method

LU decomposition is an excellent general-purpose linear systems solver but it cannot take advantage of the sparseness of a matrix. In an iterative method, if M is a good approximation to A, rapid convergence results. These observations lead to the idea of using an approximate LU factorization of A as the iteration matrix M i.e.:

$$M = L \quad = A + N \qquad 3.22$$

Where L and U are both sparse and N is small. A version of this method for symmetric matrices, known as incomplete Cholesky factorization is often used in conjunction with conjugate gradient methods. Since the matrices that arise from discretizing convection-diffusion problems or the Navier-Stokes equations are not symmetric, this method cannot be applied to them. An

asymmetric version of this method, called incomplete LU factorization or ILU, is possible but has not found widespread use. In the ILU method one proceeds as in LU decomposition but, for every element of the original matrix A that is zero, the corresponding element of L or U is set to zero. This factorization is not exact, but the product of these factors can be used as the matrix M of the iterative method. This method converges rather slowly.

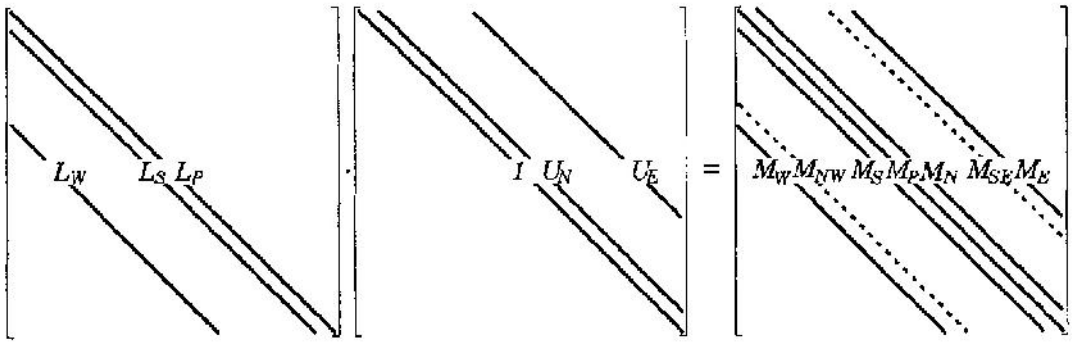


Figure 15. Schematic presentation of the matrices L and U and the product matrix M; diagonals of M not found in A are shown by dashed lines

Another incomplete lower-upper decomposition method, which has found use in CFD, was proposed by Stone (1968). This method, also called the strongly implicit procedure (SIP), is specifically designed for algebraic equations that are discretization of partial differential equations and does not apply to generic systems of equations. We shall describe the SIP method for the five-point computational molecule, i.e. a matrix with the structure shown in Fig. 3.5. The same principles can be used to construct solvers for 7-point (in three-dimensions) and 9-point (for two-dimensional non-orthogonal grids) computational molecules. As in ILU, the L and U matrices have non-zero elements only on diagonals on which A has non-zero elements. The product of lower and upper triangular matrices with these structures has more non-zero diagonals than A. For the standard five-point molecule there are two more diagonals (corresponding to nodes NW and SE or NE and SW, depending on the ordering of the nodes in the vector), and for seven-point molecules in 3D, there are six more diagonals. For the ordering of nodes used in these notes for 2D problems, the extra two diagonals correspond to the nodes NW and SE (see Table 3.2 for the correspondence of the grid indices (2, j) and the one-dimensional storage location index 1). To make these matrices unique, every element on the main diagonal of U is set to unity. Thus five sets of elements (three in L, two in U) need to be determined. For matrices of the form shown in Fig. 5.1, the rules of matrix multiplication give the elements of the product of L and U, M = LU:

$$
\begin{aligned}
M_W^l &= L_W^t \\
M_N^l &= L_W^l U_N^{l-N_j} \\
M_S^l &= L_S^l \\
M_P^l &= L_W^l U_E^{l-N_j} + L_S^l U_N^{l-1} + L_P^l \\
M_N^l &= U_N^L L_P^l \\
M_S^l &= L_S^l U_E^{L-1} \\
M_E^l &= U_E^L L_P^l
\end{aligned}
\qquad 3.23
$$

We wish to select L and U, such that M is as good an approximation to A as possible. At minimum, N must contain the two diagonals of M that correspond to zero diagonals of A, see Equation (3.23). An obvious choice is to let N have non-zero elements on only these two diagonals, and force the other diagonals of M to equal the corresponding diagonals of A. This is possible; in fact, this is the standard ILU method mentioned earlier. Unfortunately, this method converges slowly. Stone (1968) recognized that convergence can be improved by allowing N to have non-zero elements on the diagonals corresponding to all seven nonzero diagonals of LU. The method is most easily derived by considering the vector M :

$$(M)_p = M_p\phi_p + M_S\phi_S + M_N\phi_N + M_E\phi_E + M_W\phi_W + M_N\ \phi_N + M_S\ \phi_S \qquad 3.24$$

The last two terms are the 'extra' ones. Each term in this equation corresponds to a diagonal of M = LU. The matrix N must contain the two 'extra' diagonals of M, and we want to choose the elements on the remaining diagonals so that $N \approx 0$ or, in other words,

$$N_p\phi_p + N_S\phi_S + N_N\phi_N + N_E\phi_E + N_W\phi_W + M_N\ \phi_N + M_S\ \phi_S \approx 0 \qquad 3.25$$

This requires that the contribution of the two 'extra' terms in the above equation be nearly canceled by the contribution of other diagonals. In other words, Equation (3.25) should reduce to the following expression:

$$M_N\ (\phi_N - \phi_N^{\square}) + M_S\ (\phi_S - \phi_S^{\square}) \approx 0 \qquad 3.26$$

where $\phi_N^{\square}$ and $\phi_S^{\square}$ are approximations to ϕ_N and ϕ_S . Stone's key idea is that, since the equations approximate an elliptic partial differential equation, the solution can be expected to be smooth. This being so, $\phi_N^{\square}$ and $\phi_S^{\square}$ can be approximated in terms of the values of ϕ at nodes corresponding to the diagonals of A. Stone proposed the following approximation:

$$\begin{aligned} \phi_N^{\square} &\approx \alpha(\phi_W + \phi_N - \phi_P) \\ \phi_S^{\square} &\approx \alpha(\phi_S + \phi_E - \phi_P) \end{aligned} \qquad 3.27$$

If α = 1, these are second order accurate interpolations but Stone found that stability requires α < 1. These approximations are based on the connection to partial differential equations and make little sense for generic algebraic equations.

If these approximations are substituted into Equation (3.26) and the result is equated to Equation (3.25), we obtain all elements of N as linear combinations of MNW and MSE The elements of M, Equation (3.23), can now be set equal to the sum of elements of A and N. The resulting equations are not only sufficient to determine all of the elements of L and U, but they can be solved in sequential order beginning at the southwest corner of the grid:

$$\begin{aligned} L_W^l &= A_W^l/\left(1 + \alpha U_N^{l-N_j}\right) \\ L_S^l &= A_S^l/\left(1 + \alpha U_E^{l-1}\right) \\ L_p^l &= A_p^l/\left(L_W^l U_N^{1-N_j} + L_S^l U_S^{l-1}\right) - L_W^l U_E^{1-N_j} - L_S^l U_N^{1-N_j} \\ L_W^l &= \left(A_N^l + \alpha L_W^l U_N^{l-N_j}\right)/L_p^l \\ U_E^l &= \left(A_e^l + \alpha L_S^l U_E^{l-1}\right)/L_p^l \end{aligned} \qquad 3.28$$

The coefficients must be calculated in this order. For nodes next to boundaries, any matrix element that carries the index of a boundary node is understood to be zero. Thus, along the west boundary (i = 2), elements with index 1 - N_j are zero; along the south boundary (j = 2), elements with index 1 - 1 are zero; along the north boundary (j = N_j - I), elements with index l + 1 are zero; finally, along the east boundary (i = Ni - I), elements with index 1 + N_j are zero.

We now turn to solving the system of equations with the aid of this approximate factorization. The equation relating the update to the residual is (see Equation (3.18)):

$$L \quad \delta^{n+1} = \rho^n \qquad 3.29$$

The equations are solved as in in generic LU decomposition. Multiplication of the above equation by L^{-1} leads to:

$$U\delta^{n+1} = L^{-1}\rho^n = R^n \qquad 3.30$$

R^n is easily computed:

$$R^l = \left(\rho^l - L_S^l R^{l-1} - L_W^l R^{1-N_j}\right)/L_p^l \qquad 3.31$$

This equation is to be solved by marching in the order of increasing l. When the computation of R is complete, we need to solve Equation (3.30):

$$\delta^l = R^l - U_N^l \delta^{l+1} - U_E^l \delta^{1-N_j} \qquad 3.32$$

in order of decreasing index 1.

In the SIP method, the elements of the matrices L and U need be calculated only once, prior to the first iteration. On subsequent iterations, we need calculate only the residual, then R and finallyδ, by solving the two triangular systems. Stone's method usually converges in a small number of iterations. The rate of convergence can be improved by varying a from iteration to iteration (and point to point). These methods converge in fewer iterations but they require the factorization to be redone each time a: is changed. Since computing L and U is as expensive as an iteration with a given decomposition, it is usually more efficient overall to keep a: fixed. Stone's method can be generalized to yield an efficient solver for the nine-diagonal matrices that arise when compact difference approximations are applied in two dimensions and for the seven-diagonal matrices that arise when central differences are used in three dimensions. Unlike other methods, Stone's method is both a good iterative technique in its own right and a good basis for conjugate gradient methods (where it is called a preconditioner) and Multigrid methods (where it is used as a smoother).

1.10.4. Conjugate Gradient Method

In this section, we present a class of methods based on techniques for solving non-linear equations. Non-linear solvers can be grouped into two broad categories: Newton-like methods and global methods. The former converge very quickly if an accurate estimate of the solution is available but may fail catastrophically if the initial guess is far from the exact solution. 'Far' is a relative term; it is different for each equation. One cannot determine whether an estimate is 'close enough' except by trial and error. Global methods are guaranteed to find the solution (if one exists) but are not very fast. Combinations of the two types of methods are often used; global methods are used initially and followed by Newton-like methods as convergence is approached. Many global methods are descent methods. These methods begin by converting the original system of equations into a minimization problem. Suppose that the set of equations to be solved is given by Equation ($A \quad = Q$) and that the matrix A is symmetric and its eigenvalues are positive; such a matrix is called ***positive definite***. (Most matrices associated with problems in fluid mechanics are not symmetric or positive definite so we will need to generalize this method later.) For positive definite matrices, solving the system of equations ($A \quad = Q$) is equivalent to the problem of finding the minimum of

$$F = \frac{1}{2}\phi^T A \quad - \phi^T Q = \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} A_i \ \phi_i\phi_j - \sum_{i=1}^{n}\phi_i Q_i \qquad 3.33$$

with respect to all the ϕ_i; this may be verified by taking the derivative of F with respect to each variable and setting it equal to zero. A way to convert the original system into a minimization problem that does not require positive definiteness is to take the sum of the squares of all of the equations but this introduces additional difficulties. The oldest and best known method for seeking the minimum of a function is steepest descents. The function F may be thought of as a surface in (hyper)-space. Suppose we have some starting guess which may be represented as a point in that (hyper)-space. At that point, we find the steepest downward path on the surface; it lies in the direction opposite to the gradient of the function. We then search for the lowest point on that line. By construction, it has a lower value of F than the starting point; in this sense, the new estimate is closer to the solution. The new value is then used as the starting point for the next iteration and the process is continued until it converges. Unfortunately, while it is guaranteed to converge, steepest descents often converges very slowly. If the contour plot of the magnitude of the function F has a narrow valley, the method tends to oscillate back and forth across that valley and many steps may be required to find the solution. In other words, the method tends to use the same search directions over and over again. Many improvements have been suggested. The easiest ones require the new search directions to be as different from the old ones as possible. Among these is the conjugate gradient method. We shall give only the general idea and a description of the algorithm here. The conjugate gradient method is based on a remarkable discovery: it is possible to minimize a function with respect to several directions simultaneously while searching in one direction at a time. This is made possible by a clever choice of directions. We shall describe this for the case of two directions; suppose we wish to find values of α_1 and α_2 in

$$\phi = \phi^0 + \alpha_1 p^1 + \alpha_2 p^2 \qquad 3.34$$

Which minimize F; that is, we try to minimize F in the $p^1 - p^2$ plane. This problem can be reduced to the problem of minimizing with respect to p^1 and p^2 individually provided that the two directions are conjugate in the following sense:

$$p^1.\ Ap^2 = 0 \qquad 3.35$$

This property is akin to orthogonality; the vectors p^1 and p^2 are said to be conjugate with respect to the matrix A, which gives the method its name. This property can be extended to any number of directions. In the conjugate gradient method, each new search direction is required to be conjugate to all the preceding ones. If the matrix is non-singular, as is the case in nearly all engineering problems, the directions are guaranteed to be linearly independent. Consequently, if exact (no round-off error) arithmetic were employed, the conjugate gradient method would converge exactly when the number of iterations is equal to the size of the matrix. This number can be quite large and, in practice, exact convergence is not achieved due to arithmetic errors. It is therefore wiser to regard the conjugate gradient method as an iterative method. While the conjugate gradient method guarantees that the error is reduced on each iteration, the size of the reduction depends on the search direction. It is not unusual for this method to reduce the error only slightly for a number of iterations and then find a direction that reduces the error by an order of magnitude or more in one iteration. It can be shown that the rate of convergence of this method depends on the condition number k of the matrix where

$$k = \frac{\lambda_m}{\lambda_m} \qquad 3.36$$

and λ_m and λ_m , are the largest and smallest eigenvalues of the matrix. The condition numbers of matrices that arise in CFD problems are usually approximately the square of the maximum number of grid points in any direction. With 100 grid points in each direction, the condition number should be about 10^4 and the standard conjugate gradient method would converge slowly. Although the conjugate gradient method is significantly faster than steepest descents for a given condition number, this basic method is not very useful.

This method can be improved by replacing the problem whose solution we seek by another one with the same solution but a smaller condition number. For obvious reasons, this is called preconditioning. One way to precondition the problem is to pre-multiply the equation by another (carefully chosen) matrix. As this would destroy the symmetry of the matrix, the preconditioning must take the following form:

$$C^{-1}AC^{-1}C = C^{-1}Q \qquad 3.37$$

The conjugate gradient method is applied to the matrix $C^{-1}AC^{-1}$ i.e. to the modified problem (1.60). If this is done and the residual form of the iterative method is used, the following algorithm results (for a detailed derivation, see Golub and van Loan, 1990). In this description, ρ^k is the residual at the k[th] iteration, ρ^k is the k[th] search direction, z^k is an auxiliary vector and α^k and β^k are parameters used in constructing the new solution, residual, and search direction. The algorithm can be summarized as follows:

- Initialize by setting: $k = 0$, $\phi^0 = \phi_{ti}$, $\rho^0 = Q - A\phi_{ti}$, $p^0 = 0$, $S_0 = 10^3$
- Advance the counter: $k = k + 1$
- Solve the system: $Mz^k = \rho^{k-1}$
- Calculate:
    - $s^k = \rho^{k-1}.z^k$
    - $\beta^k = s^k/s^{k-1}$
    - $p^k = z^k + \beta^k p^{k-1}$
    - $\alpha^k = s_k/(p^k.Ap^k)$
    - $\phi^k = \phi^{k-1} + \alpha^k p^k$
    - $\rho^k = \rho^{k-1} - \alpha^k A^k$
- Repeat until convergence

This algorithm involves solving a system of linear equations at the first step. The matrix involved is $M = C^{-1}$ where C is the pre-conditioning matrix, which is in fact never actually constructed. For the method to be efficient, M must be easy to invert. The choice of M used most often is incomplete Cholesky factorization of A but in tests it was found that if M = *LU* where *L* and *U* are the factors used in Stone's SIP method, faster convergence is obtained.

1.10.5. Biconjugate Gradients and CGSTAB

The conjugate gradient method given above is applicable only to symmetric systems; the matrices obtained by discretizing the Poisson equation are often symmetric (examples are the heat conduction equation and pressure or pressure-correction). To apply the method to systems of equations that are not necessary symmetric (for example, any convection/diffusion equation), we need to convert an asymmetric problem to a symmetric one. There are a couple of ways of doing this of which the following is perhaps the simplest. Consider the system:

$$\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \psi \\ \phi \end{pmatrix} = \begin{pmatrix} Q \\ 0 \end{pmatrix} \qquad 3.38$$

This system can be decomposed into two subsystems. The first is the original system; the second involves the transpose matrix and is irrelevant. (If there were a need to do so, one could solve a system of equations involving the transpose matrix at little extra cost.) When the pre-conditioned conjugate gradient method is applied to this system, the following method, called biconjugate gradients, results:

- Initialize by setting: $k = 0$, $\phi^0 = \phi_{in}$, $\rho^0 = Q - A\phi_{in}$, $\bar{\rho}^0 = Q - A^T\phi_{in}$, $p^0 = \bar{p}^0 = 0$, $S_0 = 10^3$
- Advance the counter: $k = k + 1$
- Solve the system: $Mz^k = \rho^{k-1}$, $M^T\bar{z}^k = \bar{\rho}^{k-1}$
- Calculate:
  - $s^k = \bar{\rho}^{k-1}.z^k$
  - $\beta^k = s^k/s^{k-1}$
  - $p^k = z^k + \beta^k p^{k-1}$
  - $\bar{p}^k = \bar{z}^k + \beta^k \bar{p}^{k-1}$
  - $\alpha^k = s_k/(\bar{p}^k.Ap^k)$
  - $\phi^k = \phi^{k-1} + \alpha^k p^k$
  - $\rho^k = \rho^{k-1} - \alpha^k A^k$
  - $\bar{\rho}^k = \bar{\rho}^{k-1} - \alpha^k A^T\bar{p}^k$
- Repeat until convergence

The above algorithm was published by Fletcher (1976). It requires almost exactly twice as much effort per iteration as the standard conjugate gradient method but converges in about the same number of iterations. It has not been widely used in CFD applications but it appears to be very robust (meaning that it handles a wide range of problems without difficulty).

Given below the CGSTAB algorithm without formal derivation:

- Initialize by setting: $k = 0$, $\phi^0 = \phi_{in}$, $\rho^0 = Q - A\phi_{in}$, $u = p^0 = 0$
- Advance the counter: $k = k + 1$, and calculate:
  - $\beta^k = \rho^0 . \rho^{k-1}$
  - $\omega^k = (\beta^k\gamma^{k-1})/(\alpha^{k-1}\beta^{k-1})$
  - $p^k = \rho^{k-1} + \omega^k(p^{k-1} - \alpha^{k-1}u^{k-1})$
- Solve the system: $Mz^k = \rho^k$
- Calculate:
  - $u^k = A$
  - $\gamma^k = \beta^k/(u^k.\rho^0)$
  - $\omega = \rho^{k-1} - \gamma^k u^k$
- Solve the system: $M = \omega$
- Calculate:
  - $v = A$
  - $\alpha^k = (v.\rho^k)/(v.v)$
  - $\phi^k = \phi^{k-1} + \gamma^k z + \alpha^k y$
  - $\rho^k = \omega - \alpha^k v$
- Repeat until convergence

Note that u, v, w, y and z are auxiliary vectors and have nothing to do with the velocity vector or the coordinates y and z.

1.11. Pressure-Velocity coupling

For incompressible flows, in order to solve the governing equations (continuity and momentum), there has to be a method for coupling the two flow field variables which are pressure and velocity. There are numerous approaches, however the Semi-Implicit Method for Pressure Linked Equations (SIMPLE) is the most common approach to achieve that. It depends on an iterative approach to satisfy both governing equations. This method or algorithm repeats in every iteration. The four steps of this algorithm are shown in the flowchart of figure 16. The initial guess is ignored after the first iteration, and replaced by the (*) values from previous iterations.

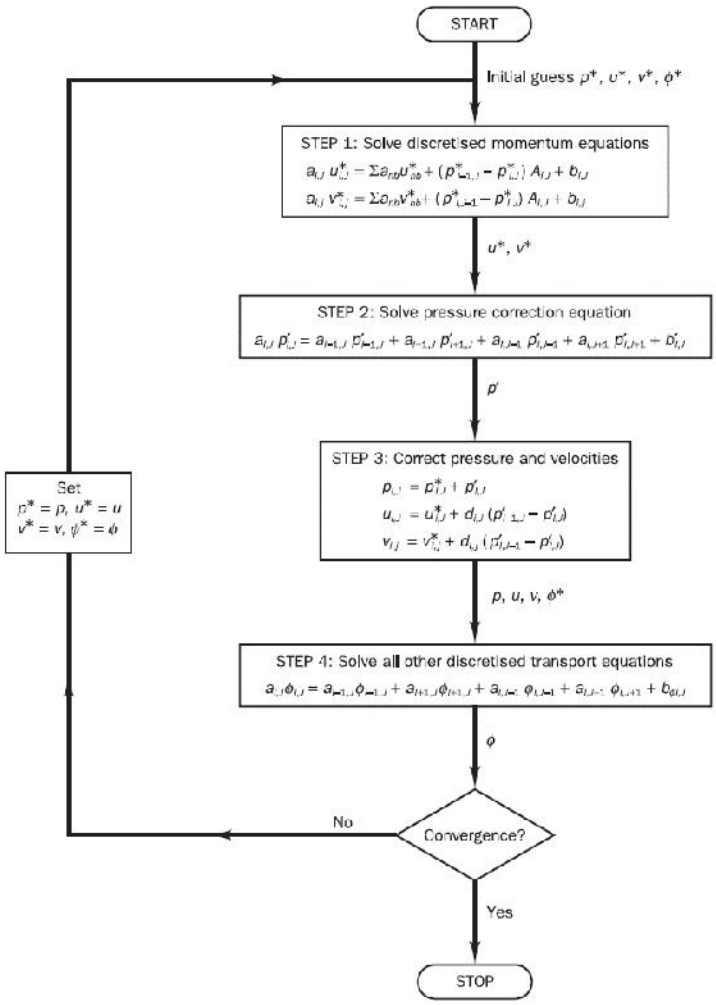


Figure 16. Flowchart of the SIMPLE algorithm loop

1.12. Under relaxation of flow field variables

At each iteration, at each cell, a new value for variable ¤ in cell P can then be calculated from that equation. It is common to apply relaxation as follows:

$$¤_P^{new,\,used} = ¤_P^{old} + U(¤_P^{new,\,predicted} - ¤_P^{old})$$

where $\phi_P^{n,u}$ is the new value used for a consecutive iteration if convergence check is not satisfied, ϕ_P^{o} is the flow field variable from the previous iteration, *U* is the relaxation factor, and $\phi_P^{n,p}$ is the value resulting from the present iteration.

Here U is the relaxation factor:

- U < 1 is under-relaxation. This may slow down speed of convergence but increases the stability of the calculation, i.e. it decreases the possibility of divergence or oscillations in the solutions.
- U = 1 corresponds to no relaxation. One uses the predicted value of the variable.
- U > 1 is over-relaxation. It can sometimes be used to accelerate convergence but will decrease the stability of the calculation.

1.13. Solution convergence

The iterative process is repeated until the change in the variable from one iteration to the next becomes so small that the solution can be considered converged. At convergence:

- All discrete conservation equations (momentum, energy, etc.) are obeyed in all cells to a specified tolerance.
- The solution no longer changes with additional iterations.
- Mass, momentum, energy and scalar balances are obtained.

Residuals measure imbalance (or error) in conservation equations. The absolute residual at point P is defined as: $R_P = \left| a_P ¤_P - \sum_{nb} a_{nb} ¤_{nb} - b \right|$. Residuals are usually scaled relative to the local value of the property ¤ in order to obtain a relative error:

$$R_{P,scaled} = \frac{\left| a_P ¤_P - \sum_{nb} a_{nb} ¤_{nb} - b \right|}{\left| a_P ¤_P \right|} \qquad (3.39)$$

They can also be normalized, by dividing them by the maximum residual that was found at any time during the iterative process. An overall measure of the residual in the domain is:

$$R^{¤} = \frac{\sum_{all\ cells} \left| a_P ¤_P - \sum_{nb} a_{nb} ¤_{nb} - b \right|}{\sum_{all\ cells} \left| a_P ¤_P \right|} \qquad (3.40)$$

It is common to require the scaled residuals to be on the order of 1E-3 to 1E-4 or less for convergence.

Introduction to turbulence modeling

1.14. Properties of turbulence

Most flows in engineering are turbulent: flows over vehicles (airplane, ship, train, car), internal flows (heating and ventilation, turbo-machinery), and geophysical flows (atmosphere, ocean). V(x_i, t) and p(x_i, t) are random functions of space and time, but statistically stationary flows such as steady and forced or dominant frequency unsteady flows display coherent features and are amendable to statistical analysis, i.e. time and place (conditional) averaging. RMS and other low-order statistical quantities can be modelled and used in conjunction with the averaged equations for solving practical engineering problems.Turbulent motions range in size from the width in the flow δ to much smaller scales, which become progressively smaller as the Re = Uδ/υ increases.

A universally valid definition of turbulence does not exist ! However, there is a number of properties that are generally common in all turbulent flow phenomena. These properties are:

1.14.1. Randomness and fluctuations

Turbulence is irregular, chaotic, and unpredictable. However, for statistically stationary flows, such as steady flows, can be analyzed using Reynolds decomposition.

$$u = \bar{u} + u' \qquad \bar{u} = \frac{1}{T}\int_{t_0}^{t_0+T} u\,dT \qquad \bar{u}' = 0 \qquad \bar{u'^2} = \frac{1}{T}\int_{t_0}^{t_0+T} u'^2\,dT \qquad (4.1)$$

where $\bar{u}$ = mean motion, u' = superimposed random fluctuation and $\bar{u'^2}$ = Reynolds stresses; RMS = $\sqrt{\bar{u'^2}}$. Triple decomposition is used for forced or dominant frequency flows $u = \bar{u} + u'' + u'$, where u'' = organized oscillation. This decomposition (or averaging) is applied to Navier Stokes equation in order to get the Reynolds-Averaged Navier Stokes (RANS) equation.

1.14.2. Nonlinearity

Reynolds stresses and 3D vortex stretching are direct result of nonlinear nature of turbulence. In fact, Reynolds stresses arise from nonlinear convection term after substitution of Reynolds decomposition into NS equations and time averaging.

1.14.3. Diffusion

Large scale mixing of fluid particles greatly enhances diffusion of momentum (and heat), i.e.,

Reynolds Stresses: $-\rho\overline{u'_i u'_j} \gg \overbrace{2\mu S_{ij}}^{viscous\ stress}$

Isotropic eddy viscosity: $-\overline{u'_i u'_j} = \nu_t S_{ij} - \frac{2}{3}\delta_{ij}k$

1.14.4. Vorticity/eddies/energy cascade

Turbulence is characterized by flow visualization as eddies, which varies in size from the largest L_δ (width of flow) to the smallest. The largest eddies have velocity scale U and time scale L_δ/U. The orders of magnitude of the smallest eddies (Kolmogorov scale or inner scale) are:

$$L_K = \text{Kolmogorov micro-scale} = \left[\frac{\nu^3 \varepsilon}{U^3}\right]^{\frac{1}{4}} \qquad (4.2)$$

L_K = O(mm) >> $L_{\text{mean free path}} = 6 \times 10^{-8}$ m

Velocity scale = $(\nu\varepsilon)^{1/4}$ = O(10^{-2}m/s)

Time scale = $(\nu/\varepsilon)^{1/2}$ = O(10^{-2}s)

1.14.5. Dissipation

$\ell_0 = L_\varepsilon$

$u_0 = \sqrt{k} \qquad k = \overline{u'^2} + \overline{v'^2} + \overline{w'^2}$

$= 0(U)$

$\text{Re}_\varepsilon = u_0 \ell_0 / \nu = big$

→ Energy comes from largest scales and fed by mean motion

ε = rate of dissipation = energy/time

$= u_0^2 / \tau_o \qquad \tau_o = \ell_0 / u_0$

$= u_0^3 / l_0$ independent υ $\qquad L_K = \left[\frac{\nu^3}{\varepsilon}\right]^{\frac{1}{4}}$

→ Dissipation occurs at smallest scales

1.14.6. Mean flow features

- Fluctuations are large ~ 11% U_∞
- Presence of wall causes anisotropy, i.e., the fluctuations differ in magnitude due to geometric and physical reasons. $\overline{u'^2}$ is largest, $\overline{v'^2}$ is smallest and reaches its maximum much further out than $\overline{u'^2}$ or $\overline{w'^2}$. $\overline{w'^2}$ is intermediate in value.
- $\overline{u'v'} \neq 0$ and, as will be discussed, plays a very important role in the analysis of turbulent shear flows.
- Although $\overline{u_i u_j} = 0$ at the wall, it maintains large values right up to the wall
- Turbulence extends to y > δ due to intermittency. The interface at the edge of the boundary layer is called the super-layer. This interface undulates randomly between fully turbulent and non-turbulent flow regions. The mean position is at y ~ 0.78 δ.

- Near wall turbulent wave number spectra have more energy, i.e. small λ, whereas near δ large eddies dominate.

1.15. The Reynolds Averaged Navier-Stokes (RANS) equation

1.15.1. Time averaging

For stationary flow, the mean is not a function of time and we can use time averaging. $\bar{u} = \frac{1}{T}\int_{t_0}^{t_0+t} u(t)\,dt$ where T > any significant period of $u' = u - \bar{u}$ (e.g. 1 sec. for wind tunnel and 20 min. for ocean).

1.15.2. Ensemble averaging

For non-stationary flow, the mean is a function of time and ensemble averaging is used $\bar{u}(t) = \frac{1}{N}\sum_{i=1}^{N} u^i(t)$ where N is large enough that $\bar{u}$ independent and $u^i(t)$ = collection of experiments performed under identical conditions (also can be phase aligned for same t=o).

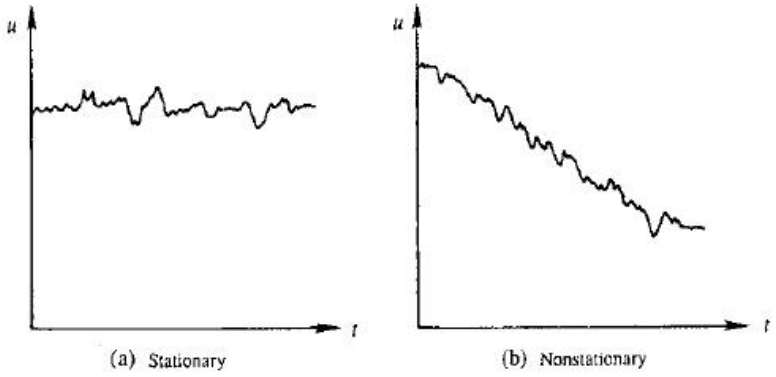


Figure 17. Stationary and non-stationary time series of a turbulent flow

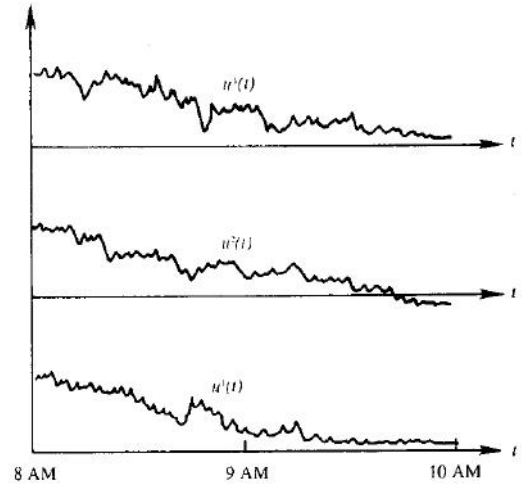


Figure 18. An ensemble of the function u(t).

1.15.3. Spatial averaging

The spatial averaging represents homogenous turbulence. It shows that the mean velocity can be expressed as: $\bar{u} = \frac{1}{V}\iiint_x \quad u(x,y,z,t)d$.

1.15.4. Derivation of the RANS equation

The RANS equation can be derived by substituting for the velocity vector by the relation $u_i(x_{i,j,k},t) = U_i(x_{i,j,k}) + \tilde{u}_i(x_{i,j,k},t)$ where U_i is the mean velocity and $\tilde{u}_i$ is the fluctuating velocity. This substitution represents the Reynolds averaging, where the mean velocity expresses statistically averaged stationary turbulence, and the fluctuating velocity expresses the non-stationary turbulence. By applying that substitution to the incompressible Navier-Stokes equation, we get:

$$\rho\left(\frac{\partial(U_i+\tilde{u}_i)}{\partial} + \frac{\partial}{\partial x_j}\left(U_iU_j + \tilde{u}_i\tilde{u}_j\right)\right) = -\frac{\partial}{\partial x_i} + \mu\frac{\partial}{\partial x_j}\left(\frac{\partial(U_i+\tilde{u}_i)}{\partial x_j} + \frac{\partial(U_j+\tilde{u}_j)}{\partial x_i}\right) + S \quad (4.3)$$

Rearranging and simplifying, we get:

$$\rho\left(\frac{\partial U_i}{\partial} + U_j\frac{\partial U_i}{\partial x_j}\right) = -\frac{\partial}{\partial x_i} + \mu\frac{\partial}{\partial x_j}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) - \rho\tilde{u}_i\tilde{u}_j + S \quad (4.4)$$

where the term $\rho\tilde{u}_i\tilde{u}_j$ is a symmetric tensor called the Reynolds stress tensor.

$$-\rho\tilde{u}_i\tilde{u}_j = -\rho\begin{bmatrix}\tilde{u}^2 & \tilde{u}\tilde{v} & \tilde{u}\tilde{w}\\ \tilde{v}\tilde{u} & \tilde{v}^2 & \tilde{v}\tilde{w}\\ \tilde{w}\tilde{u} & \tilde{w}\tilde{v} & \tilde{w}^2\end{bmatrix} \quad (4.5)$$

Closure Problem:

- RANS equations differ from the NS equations due to the Reynolds stress terms
- RANS equations are for the mean flow ; thus, represent 4 equations with 10 ten unknowns due to the additional 6 unknown Reynolds stresses
- Equations can be derived for by summing products of velocity and momentum components and time averaging, but these include additionally 10 triple product unknowns. Triple products represent Reynolds stress transport.
- Again equations for triple products can be derived that involve higher order correlations leading to fact that RANS equations are inherently non-deterministic, which requires turbulence modeling.
- Turbulence closure models render deterministic RANS solutions.
- The NS and RANS equations have paradox that NS equations are deterministic but have nondeterministic solutions for turbulent flow due to inherent stochastic nature of turbulence, whereas the RANS equations are nondeterministic, but have deterministic solutions due to turbulence closure models.

1.16. Boussinesq eddy viscosity assumption

In order to close the system of equations represented by RANS and find closure for the Reynolds stress tensor, Boussinesq proposed his eddy viscosity assumption in 1877. This assumption builds an analogy between the turbulent rate of strain due to Reynolds stress and the rate of strain due to viscous stress in laminar flow. Hence, introduces a fictitious scalar called the *eddy* or *turbulent* viscosity (μ_t). Such scalar represents the effect of energy transport due to turbulent motion on the mean flow. Hence, the assumption was derived as:

$$-\rho\tilde{u}_i\tilde{u}_j = \mu_t\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} - \frac{2}{3}\frac{\partial U_k}{\partial x_k}\delta_{ij}\right) - \frac{2}{3}\rho\ \delta_{ij} \tag{4.6}$$

where k is the turbulence kinetic energy, originally expressed as $k = \frac{1}{2}\overline{u_i^2} = \frac{1}{2}\left(\overline{u^2} + \overline{v^2} + \overline{w^2}\right)$ and δ_{ij} is Kronecker delta, $\delta_{ij} = 0\ if\ i \neq j, 1\ if\ i = j$.

1.17. Most common eddy viscosity turbulence models

An eddy viscosity turbulence model is a set of equations that aim at calculating μ_t as a function of the mean flow field variables. There are several models which include once, two, three and four equation turbulence models. However, the most common eddy viscosity models, which are capable of predicting wide range of engineering flows, belong to the $k-\varepsilon$ and $k-\omega$ families. Here are the transport equations for these models:

1.17.1. The standard $k-\varepsilon$ model

Turbulence kinetic energy (k) equation:

$$\rho\left[\frac{\partial k}{\partial t} + V\nabla\cdot k\right] = \nabla\cdot\left[\left(\mu + \frac{\mu_T}{\sigma_k}\right)\nabla k\right] + \mu_T S^2 - \rho\varepsilon \tag{4.7}$$

Dissipation rate equation:

$$\rho\left[\frac{\partial\varepsilon}{\partial t} + V\nabla\cdot\varepsilon\right] = \nabla\cdot\left[\left(\mu + \frac{\mu_T}{\sigma_\varepsilon}\right)\nabla\varepsilon\right] + C_{\varepsilon 1}\frac{\varepsilon}{k}\mu_T S^2 - C_{\varepsilon 2}\rho\frac{\varepsilon^2}{k} \tag{4.8}$$

Eddy viscosity:

$$\mu_T = \rho C_\mu \frac{k^2}{\varepsilon} \tag{4.9}$$

where $C_{\varepsilon 1}$=1.44, $C_{\varepsilon 2}$=1.92, C_μ=0.09, σ_k=1.0, σ_ε=1.3

1.17.2. The realizable $k-\varepsilon$ model

Turbulence kinetic energy (k) equation is the same as equation (4.7), while specific dissipation rate equation is:

$$\rho\left[\frac{\partial\varepsilon}{\partial t} + V\nabla\cdot\varepsilon\right] = \nabla\cdot\left[\left(\mu + \frac{\mu_T}{\sigma_\varepsilon}\right)\nabla\varepsilon\right] + C_1\rho S_\varepsilon - C_2\rho\frac{\varepsilon^2}{k + \sqrt{\nu\varepsilon}} \tag{4.10}$$

$$C_1 = \max\left(0.43, \frac{\eta}{\eta + 5}\right) \qquad C_2 = 1.9$$

while C_μ is not a constant, as in the standard $k-\varepsilon$, and it is computed from:

$$C_\mu = \frac{1}{A_0 + A_s U^* \frac{k}{\varepsilon}} \tag{4.11}$$

Where

$$A_0 = 4.0;\ U^* = \sqrt{S_{ij}S_{ij} + \tilde{\Omega}_{ij}\tilde{\Omega}_{ij}},\ A_s = \sqrt{6}\cos\left(\frac{1}{3}\arccos\left(\sqrt{6}W\right)\right),\ W = \frac{\sqrt{8}S_{ij}S_{jk}S_{ki}}{S^3}$$

and the vorticity tensor $\tilde{\Omega}_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} - \frac{\partial \bar{u}_j}{\partial x_i}\right)$

1.17.3. The standard $k-\omega$ model

The standard $k-\omega$ is based on the work of Wilcox, which is its essence an empirical model based on model transport equations for the turbulence kinetic energy (k) and the specific dissipation rate (ω). With a major advantage in its numerical stability, compared to the standard $k-\varepsilon$ model, the standard $k-\omega$ model is very sensitive to the free-stream boundary conditions. In the case of diffusion flame, this sensitivity might cause inaccurate predictions of the flame physics. This is because of the sensitive dependence of the EDM turbulent reaction rate on the predicted turbulence dissipation rate.

The Kinematic eddy viscosity equation in the standard $k-\omega$ model is:

$$\nu_T = \frac{k}{\omega}$$

and the equation for the production of turbulence kinetic energy is:

$$\frac{\partial k}{\partial t} + V\nabla \cdot k = \nabla \cdot \left[\left(\nu + \sigma^* \nu_T\right)\nabla k\right] + \tau_{ij}\nabla V - \beta^* k\omega \tag{4.12}$$

while the specific dissipation rate equation is expressed as:

$$\frac{\partial \omega}{\partial t} + V\nabla \cdot \omega = \nabla \cdot \left[\left(\nu + \sigma \nu_T\right)\nabla \omega\right] + \alpha \frac{\omega}{k}\tau_{ij}\nabla V - \beta \omega^2 \tag{4.13}$$

where $\sim = \frac{5}{9}$, $= \frac{3}{40}$, $^{*} = \frac{9}{100}$, $\pm = \pm^{*} = \frac{1}{2}$, and the dissipation rate is expressed as $£ = {}^{*}\mu k$

Verification and Validation in CFD

CFD produces results that resembles the solution of governing and transport equations based on initial and boundary conditions. These equations, by definition, are highly non-linear, therefore they have sensitive dependence on initial and boundary conditions. A slight alteration in such conditions may produce a solution deviation from the actual physical conditions. In addition, the numerical schemes used to solve the equations contains a number of schemes and stages that introduce errors to the final solution of the systems of equations. These errors such as truncation and discretization errors results in some sort of dependence between the implemented numerical solution parameters and the CFD simulation results. Therefore, the reliability of CFD simulations must be evaluated systematically in order to apprehend the errors in the simulation in one hand, and appreciate the value of the results within the engineering design and optimization process in the other hand.

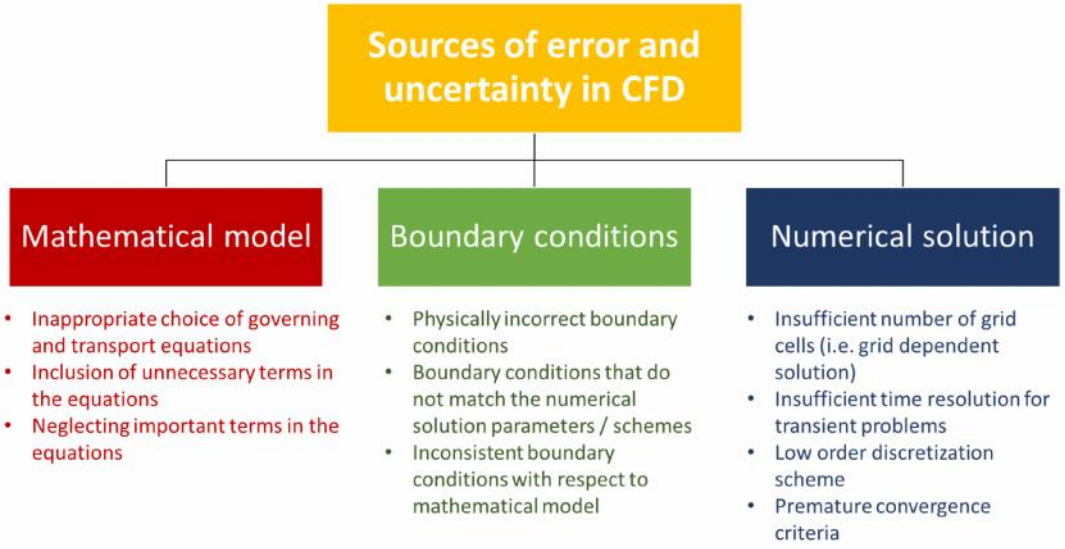


Figure 19. Sources of error in CFD

The verification and validation (V&V) process aims at identifying, quantifying and minimizing the errors in a CFD simulation. Without proper V&V practice, the CFD simulation results become of no scientific value. There are numerous approaches of practicing V&V. Here, a sequential approach, based on the author's experience, is proposed. This approach is schematically illustrated in figure 20. The verification process aims at investigating if the equations are solved in a correct manner, while the validation process investigates if the correct equations are solved or not.

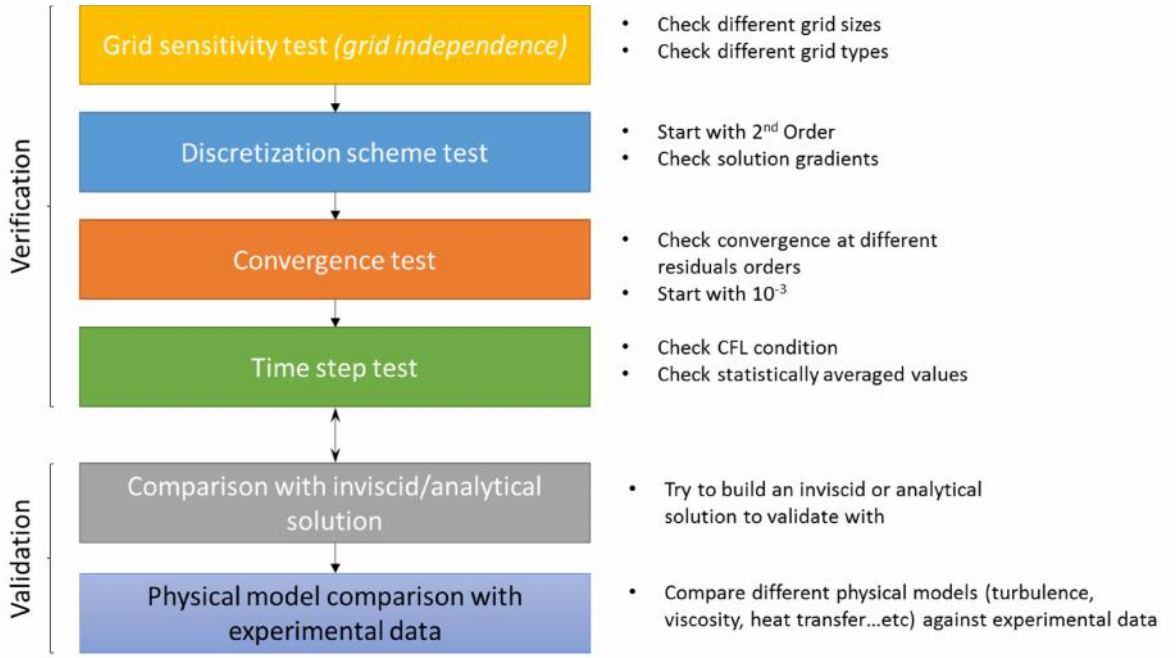


Figure 20. Verification and Validation process flowchart

Bibliography

Henk Kaarle Versteeg, Weeratunge Malalasekera (2007) An Introduction to Computational Fluid Dynamics: The Finite Volume Method, Pearson Education Limited

Joel H. Ferziger, Milovan Peric (2012) Computational Methods for Fluid Dynamics, Springer Science

David C. Wilcox (1994) Turbulence Modeling for CFD, DCW Industries