Numerical Techniques for Process Modeling
Finite Volume Method
and
Computational Fluid Dynamics

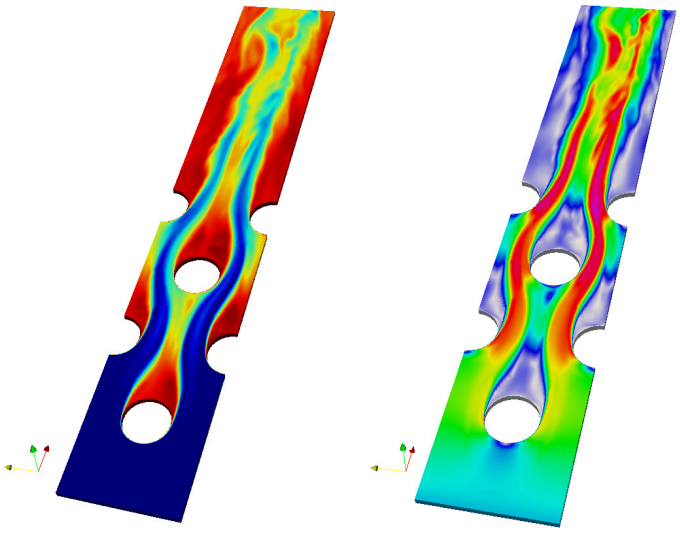
KEB-45250
Numerical Techniques for Process Modelling,
Prosessien numeerinen mallinnus
5 credits

Antti Mikkonen
antti.mikkonen@iki.fi
Tampere University of Technology

Version 0.3

Updated on:

February 20, 2018



License

Cover picture by Turo Välikangas.

Publication version history

These notes are written during the first implementation of this course and published in parts. Backward compatibility in equation numbers etc. is attempted. Below you'll see the main changes between published versions. Small modifications are not noted.

Version	Date	Added	Changes	Notes
0.1	06.02.2018	Sections 1-2		
0.2	13.02.2018	Section 3	Changed figure numbering scheme	
0.3	20.02.2018	Section 4		

Preface

This lecture note is meant for students on the course Numerical Techniques for Process Modelling, arranged on Tampere University of Technology. The course consist of three parts: computational fluid dynamics, general use of numerical methods in energy related industrial problems, and reaction modeling. This booklet considers the computational fluid dynamics part.

This lecture note is a custom fitted to contain the material considered on the course and presented in my style. If you prefer a full text book, see the introductory level text book by Versteeg and Malalasekera, An introduction to Computational Fluid Dynamics [6]. More detailed pointers will be given in the text.

This is the first time this lecture note is used and the lecture note will be written as the course progresses. Errors are inevitable. Feel free to email the author: antti.mikkonen@iki.fi.

Very few things in this world are solo achievements. Neither are these notes. Many of the used pictures are freely available in the Internet and the sources are listed in the references. People who have contributed in some less obvious way are listed in the next section. Thanks for all the contributers!

Contributions

Following is a more or less chronological and probably incomplete list of people who have contributed to these notes by comments, error corrections, etc.

Turo Välikangas
Niko Niemelä
Mikko Pirttiniemi
Elli Heikkinen

Acronyms and translations

CFD	Computational fluid dynamics	Virtauslaskenta
FEM	Finite element method	Elementtimenetelmä
RANS	Reynolds-averaged Navier-Stokes (equations)	
NS	Navier-Stokes (equations)	
FVM	Finite volume method	Tilavuusmenetelmä
DNS	Direct numerical simualtion	

Contents

1 Introduction

This section given an overview of some of the key concepts of CFD with very little theoretical detail. Most subject will be considered in more detail later.

Computational fluid dynamics (CFD) offers great flexibility for an engineer and can offer valuable insight into complex problems. Industrial machines often have a complex geometry well beyond the scope of analytical methods. For complex phenomena in industrial scale, such as airplane or power plant design, CFD is often the only viable option as a full scale experiment would be too expensive.

A word of warning is, however, in order.

The acronym of Computational Fluid Dynamics, CFD, has many interpretation: Colorful Fluid Dynamics, Colors For Directors, Cleverly Forged Data, etc. As usual, there is some truth in the joke. It is very easy to produce good looking false results with CFD. Learning how to use CFD programs well enough to produce pretty videos may give false confidence. With few exceptions, CFD programs need an expert user to be reliable.

1.1 Mesh Basics

Computational fluid dynamics (CFD) studies fluid flow using computers to solve numerical approximations of often complex problems. There are many ways to build the approximations, but in this text we will study exclusively finite volume method (FVM). In FVM we divide the calculation domain in to small volumes called cells. The cells form a mesh that fill the computational domain, see Fig. 1.1. A typical mesh contains hundreds of thousands or millions of cells.

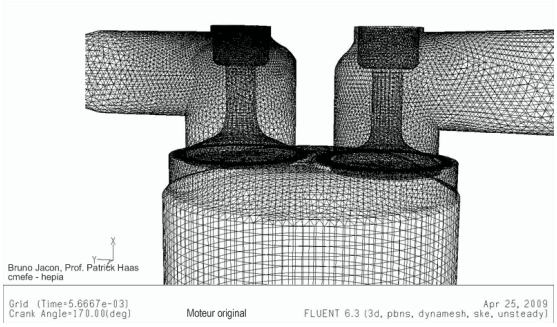


Figure 1.1: Engine mesh[2]

Mesh might be the most important concept in finite volume method. Each cell form a control volume, that is very similar to the control volume used in analytical fluid dynamics. The main difference is that in finite volume method the control volume has a finite size. When solving differential equations analytically we would typically let the control volume be infinitesimally small.

In finite volume method we assume that a single value of a field variable, say pressure, is enough to represent the value of that field inside the whole cell. In other words, the original continuous field is approximated with a discrete, non-continuous one.

As the mathematical concepts may quickly become a little confusing without a concrete example, let us consider a fully developed turbulent mean velocity profile in a pipe, see Fig. 1.2. The velocity distribution is, of course, continuous. Discretizing the profile with ten cells gives the discretized profile in Fig. 1.2. This is analogous to using a discretized numerical solution. In the example case with ten equally sized cells, using finite volume method would probably fail miserably as the mesh is unable to represent the fast change in velocity profile near the wall. We would need smaller cells near the wall. In real use cases the mesh is usually refined near walls and other regions where fast changes happen, see Fig. 1.3.

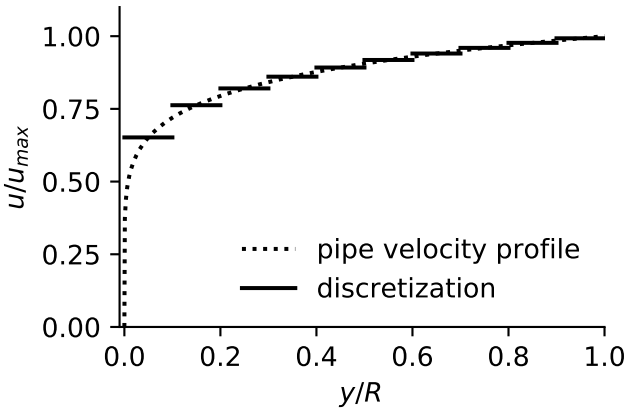


Figure 1.2: Fully developed turbulent mean velocity profile in a pipe

The guiding principle of meshing is to have small cells where fast change occurs and larger cells elsewhere to avoid unnecessary computational cost. In addition, we want our cells to have regular shapes and the cell face normals to align with stream lines. This allows for better numerical accuracy.

In Fig. 1.3 we see two main strategies to achieve these goals. On the left we have a structured mesh. The mesh is pieced together solely from morphed rectangles. For simple shapes this strategy often lead to high quality meshes if enough effort is spend on the process.

On the right in Fig. 1.3 we see an unstructured mesh. Unstructured meshes have more chaotic structure than structured ones. As a rule of a thump, one can usually get a better mesh using structured meshes if enough effort is spend in the meshing process. Unstructured meshes are, however, a lot easier and faster to generate. Structured meshes have to be build manually but for unstructured meshes there are many automated algorithms. For complex shapes, unstructured meshes are practically the only option. The two strategies can be combined. One could, for example, use a structured mesh near the wing in Fig. 1.3 and unstructured one elsewhere.

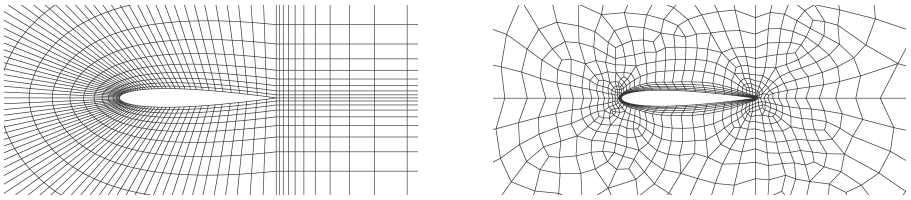


Figure 1.3: Structured and unstructured (low quality) meshes of a 2D wing

The cells in meshes may have an arbitrary shape. Simple, regular shapes like the ones in Fig. 1.4 are, however, preferred for numerical accuracy. In finite volume method the cell shape and aliment to stream lines is one of the most important factor for reliable solution. In practice, however, it is often more convenient to use a brute force approach. Even low quality cells give good results if small enough cells are used. The computational expense limits this approach to meshing.

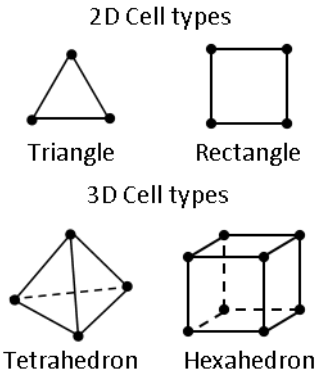


Figure 1.4: Cell types

Now that we have learned some basics of meshes, let us study the other aspects of CFD as well.

1.2 Steps of CFD

We started our discussion about CFD with the basics of meshing. Now we will quickly familiarize ourselves with the other steps of CFD.

The process of CFD is usually divided in three parts:

- Preprocessing
- Solver
- Post-processing

Simply put, preprocessing is everything that is done before opening the actual CFD solver programs. Solver part happens in the CFD solver. Post-processing is everything that happens after you close the actual solver. It is common to use different programs for each of these tasks.

1.2.1 Preprocessing

Preprocessing consist of two main parts: CAD and meshing. A CFD solver operates in a mesh. Mesh is usually based on a geometry created in a CAD program. Some meshing strategies do not need a CAD geometry, but in general the two are closely linked.

In order to make a mesh with a suitable level of detail, a CAD model with suitable level of detail is needed. The suitable level of detail depends on a case. The main idea is that the CAD geometry and the mesh should contain the features that are important for the problem at hand and nothing else.

The choice of level of detail in the model is far from easy. Experience and intuition are needed. If possible, it's a good practice to look up a scientific paper on a similar case or ask from someone with experience.

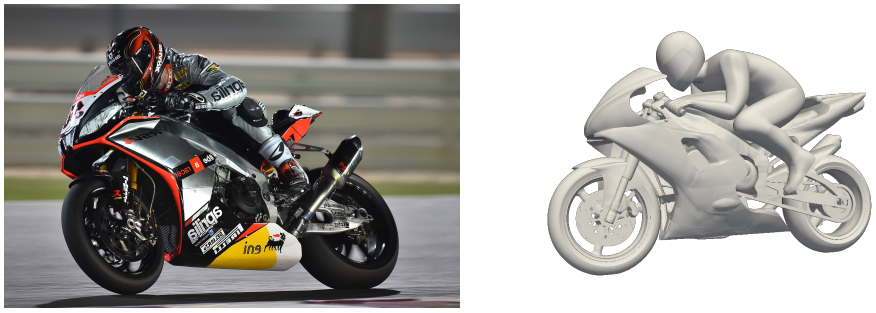


Figure 1.5: Geometry simplification[7]

In Fig. 1.5 we see a photo of a motorcycle and a simplified CAD model of the motorcycle surface. If we are modeling the flow outside the motorcycle we only want to include the outer shell of the motorcycle in our CAD model. Many details on the outer surface may also be ignored.

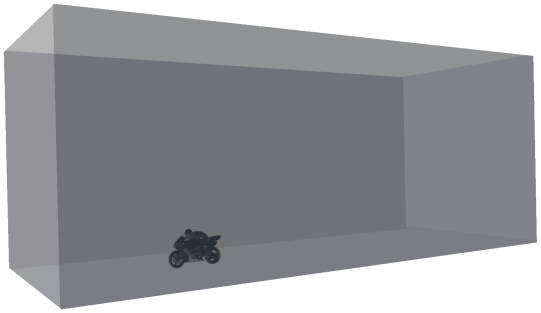


Figure 1.6: Calculation domain

It is important to understand that if we want to solve the air flow outside the motorcycle, we won't be modeling the motorcycle itself. Our model will consist of the air around the motorcycle. In Fig. 1.6 we see a typical calculation domain for an outside flow of a vehicle. We need to be able to solve all the

important flow phenomena and therefore our domain must be large enough to contain them.

The flow in front of the motorcycle is mostly unaffected by the motorcycle expect for a short distance in front of the motorcycle. Therefore we only model a small amount of air in front of the motorcycle.

The motorcycle will create a long wake behind it. The wake is important for the aerodynamics of the motorcycle and we want to include it in our model. Therefore a large amount of air is modeled behind the motorcycle.

The air flow on top of the bike and to it sides will be mostly unaffected by the motorcycle until the flow reaches the wake region. The width and height must be suitably sized to contain the wake.

Modeling a unnecessarily large domain is computationally expensive and will force us to use larger cells. Again, experience and understanding of the flow phenomena is needed.

Setting boundary conditions is usually considered as part of the solver setup, see Fig. 1.9. However, the used boundary conditions directly affect the used domain and mesh. It is quite common that there is only a rough knowledge of the boundary condition. It could, for example, be that the total inlet mass flow is known but the velocity profile is not known. In such cases it is convenient to extend the inlet region far upstream to allow the velocity profile to develop before it reaches the region of interest.

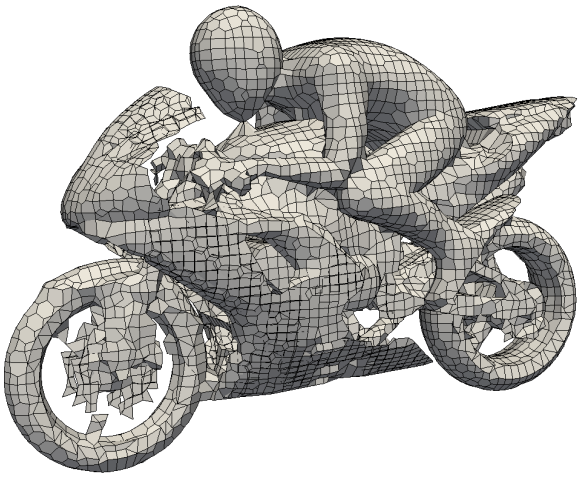


Figure 1.7: Surface mesh (low quality)

The mesh must have sufficiently small cells to accurately represent the modeled phenomena. In Fig. 1.7 there are some obvious problems. Half of the disk brake, for example, has been cut out of the simulation due to too coarse a mesh.

Mesh quality is one of the most important parts of practical CFD. Too coarse a mesh and the model won't be able to represent reality. Too fine a mesh and the required computational resources grow too large. Unfortunately, there is no way to truly know if the mesh is fine enough in a complex case. There are,

however, best practices.

A simple and effective method is just to look at the mesh. There might be some obvious errors there. Most CFD programs and meshing tools are also able to report the mesh quality.

After the solution is ready, it's a good practice to, again, have a critical look at the results. Sometimes there is something obviously wrong there. You may also notice that your initial assessment of the flow was wrong and that you have placed small cells in wrong places.

The closest one can get to assessing the mesh quality is to run a formal mesh independency test. The basic idea of a mesh independency test is to run a series of simulations with unceasingly fine meshes and see when the results won't change anymore. An example of mesh refinement is shown in Fig. 1.8.

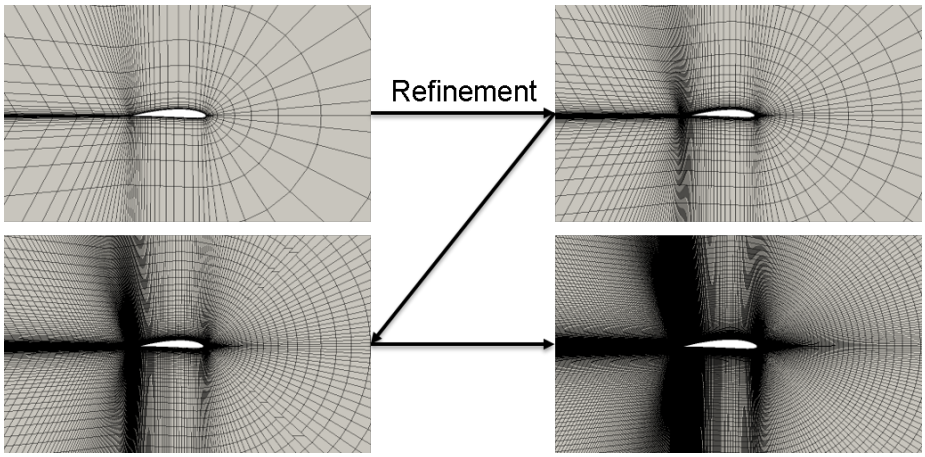


Figure 1.8: Mesh refinement

Refining the mesh with the same meshing strategy, as is done in Fig. 1.8, is relatively simple as the meshing parameters can be easily adjusted. The results from the coarser meshes can also be used as initial values to speed up the process.

A formal mesh independency test would include many different meshing strategies. Such a test a very time consuming and often impractical.

1.2.2 Solver

Understanding a CFD solver requires much more theory than the basics of preprocessing and post-processing. We will therefore only give a very short introduction here.

Solver set up contains all the physics and numerical method definitions of the problem. At this stage we choose

- steady state or unsteady simulation
- compressible flow or incompressible flow
- boundary conditions

- fluid properties
- turbulence model
- additional models like chemistry or radiation

Inside the solution domain the physical models describe the flow. At the boundary, however, additional information is needed. Typical boundary conditions are inlet velocity profiles and pressure levels.

Mathematical concepts such as symmetry are also often used to save computational results. Using symmetry requires understanding of the phenomena. For example, wakes of symmetrical obstacles are not symmetrical.

See Fig. 1.9 for examples of boundary conditions. The mathematical definitions will be given later.

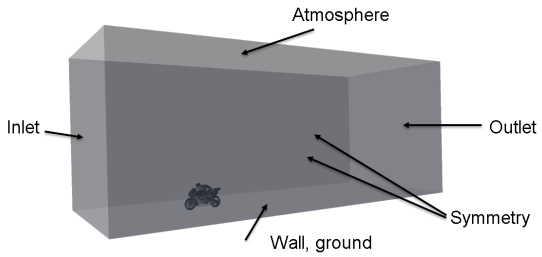


Figure 1.9: Boundary conditions

In general, fluid flow problems do not have a steady state solution. Some low Reynolds number flows are truly steady state, but usually a steady state solution is a mathematical artifact, that hopefully represent the mean flow. However, the mean properties are often what we are interested in. Solving the fine details would just cost more in computational time and make post processing more difficult. With suitable choices of of turbulence model and geometry simplification one may get a useful steady state solution of a complex flow.

An example is the well known test case by Pitz and Daily [4], see Figs. 1.10 and 1.11. The flow is turbulent, but the modeled flow still reaches a steady state solution.

Fluid flows are highly nonlinear by nature. In the context of CFD this mean that we need to an iterative process to produce a result. Iterations are needed for both steady and transient problems. As both the time steps and iterations of a steady state solution are based on previously calculated solution, they are mathematically similar. From Figs. 1.10 and 1.11 we can see the similarity.

Transient simulations are done when we are interested in the changes of flow field in time or when steady state solution is not possible. Many flows don't have steady state solutions and unsteady solution is the only choice. Due to the similarity of the time stepping and iterations, transient solvers are also used even if we are only interested steady state solution. They are usually more stable than steady state solvers and with long time steps they may have similar convergence rates.

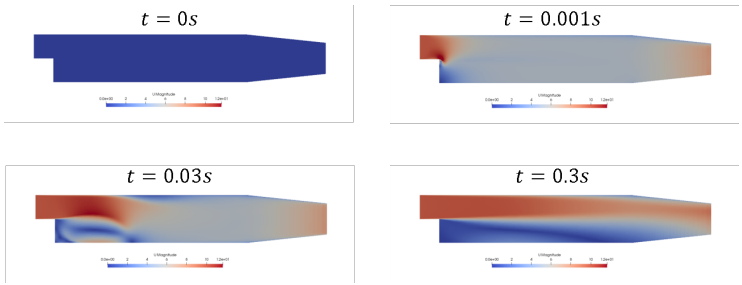


Figure 1.10: Transient solution

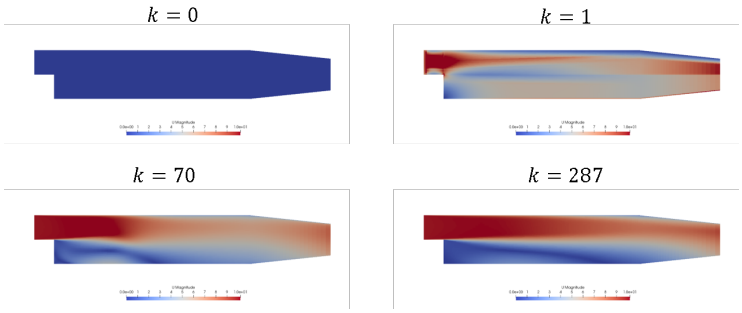


Figure 1.11: Steady solution

During solver set up, it is important to set up some monitors for you simulation. The best kind of monitors are the physical values you are interested in. In case of a heat exchanger, the values would often be mean heat transfer coefficient and pressure drop. It is often illustrative to plot those values during iterations or time steps to monitor convergence. In the example in Fig. 1.12 we have a steady state simulation of an heat exchanger. As is typical to complex flows, the steady state simulation never converges, but oscillates around a mean. This is often good enough.

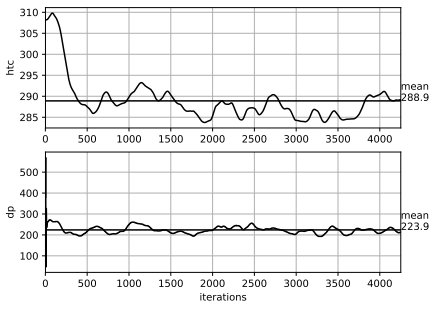


Figure 1.12: Pseudo convergence of steady state solution

Most CFD programs use a mathematical concept called residual as a convergence criteria. It is near impossible to know when the solution is good enough

from residual alone and it is recommended to always use some physical quantity instead.

1.2.3 Post-processing

Post processing is the step in which the actual value of the simulation is produced. It is critical to not only be able to produce high quality simulations, but to report the results also.

A typical CFD simulation produces gigabytes of detailed data. Typically, only one or two integral values are truly of interest. In case of the motorcycle example, the interesting value could be the drag force.

The detailed information is, however, useful for developing understanding of the flow. It's sometimes easy to spot mistakes from the solution by just looking at the field values. Similarly, it is sometimes very useful to study the qualitative behavior of the simulation. It may give insight to the phenomena that is not available from measurements. See for example the heat exchanger in Fig. 1.13. It would be next to impossible the measure the velocity and temperature profiles inside the device and visualize them.

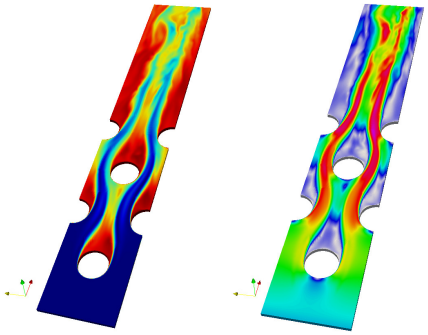


Figure 1.13: Temperature and velocity fields at one time step[5]

Colorful figures and videos are also great for marketing and explaining your results.

2 Heat Conduction (diffusion)

The topics discussed in this section may be found in a from Versteeg and Malalasekara 2007, Chapter 5. [6]

In this section we learn to implement a finite volume solver for heat conduction. This will allow us to create our own solvers for heat conduction and to understand more complex problems later on.

The goal of finite volume method (FVM), and most other CFD methods, is to divide a complex continuous problem into manageable small pieces. An approximate solution of the original problem is expressed with a large number of simple arithmetic equations. These equations are readily assembled in to a matrix and solved with a computer.

The main advantages of numerical solution over analytical solution is the ease of solving problems in complicated geometries, boundary conditions or with varying fluid properties. For such problems analytical solutions often don't exists or are very time consuming to implement. A word of warning is, however in order: it is very easy to produce false results with numerical tools! Especially with commercial software. Therefore it is important to always verify the numerical method with a similar analytical solution if such is available.

During this course we become familiar with Finite Volume Method (FVM). Other very similar numerical methods include Finite Element Method (FEM) and Finite Difference Method (FDM). In general, numerical solution of fluid related problems are called Computational Fluid Dynamics (CFD). FVM method is the preferred method for fluid flow problems because of it's simplicity and conservativeness (mass balance etc.).

2.1 Developing the equations for 1D problem

See Versteeg and Malalasekare [6] page 114, Section 4.1-4.3.

We will start with a 1D problem and only briefly consider 2D and 3D problems at the end.

The key concept for nearly all numerical solutions is discretization of the problem. In Finite Volume Method we divide the solution domain in small control volumes very similar to the ones used in analytical modeling, see Fig. 2.1. The difference is that in analytical modeling we allow to control volume to become infinitesimally small. In Finite Volume Method we use some suitably small size instead.

Together the control volumes fill the solution domain. We will derive the equations for a single cell at a time. We will start by defining notations first. In general, we need information of the cell we are studying and its immediate neighbors.

We use letter P to refer to the central point (node) of the control volume we are currently studying and W and E to refer to neighboring nodes as shown in Fig. 2.2.

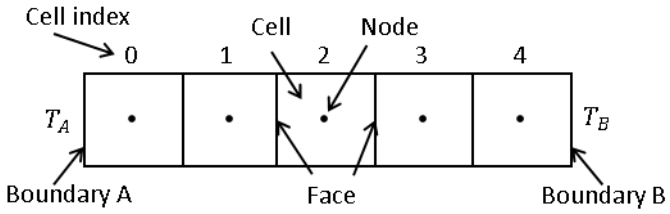


Figure 2.1: Discretization and terminology

In addition to control volumes and nodes we also need to refer to the faces at control volume boundaries. We call these faces w and e, see Fig. 2.3. The cell size, or distance from one face to another is called Δx as shown in Fig. 2.3.

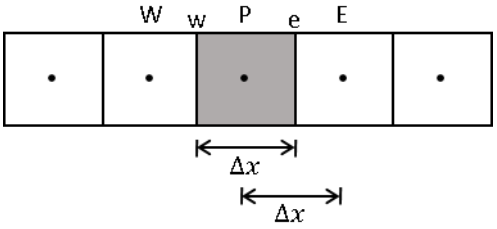


Figure 2.2: Faces and distances

As a practical example, our domain could be a 100mm long metal rod and we could divide it in 100 control volumes, each 1mm long.

Let us consider one control volume, the one around node P. If there is a heat flux density $q = -k\frac{\mathrm{d}T}{\mathrm{d}x}$ from neighboring cell W, across the face w at the boundary between nodes W and P, to cell P, the heat flux is $(qA)_w = -\left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_w$, where A_w is the face area. The locations are marked with subscripts. Similarly for the east face $(qA)_e = -\left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_e$.

If there is a mean volumetric heat source $\overline{S}$ inside the control volume, the total heat power is $S = \overline{S}\Delta V$. A practical example of such a source is electric heating.

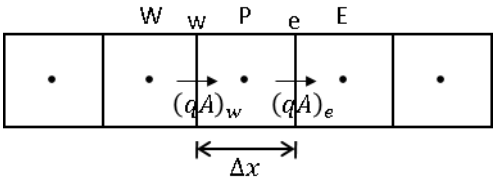


Figure 2.3: Heat conduction

Similarly to the control volume used in analytical methods, we can now collect the terms in a balance equation for heat as

$$
\begin{aligned}
&(qA)_w - (qA)_e + \overline{S}\Delta V = 0 \\
\Leftrightarrow &\left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_e - \left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_w + \overline{S}\Delta V = 0
\end{aligned} \tag{2.1}
$$

The Eq. 2.1 contains the essence of Finite Volume Method. Using a familiar control volume approach, we have discretized our problem for heat fluxes.

What remains is to express the temperature gradients $\left(\frac{\mathrm{d}T}{\mathrm{d}x}\right)_w$ and $\left(\frac{\mathrm{d}T}{\mathrm{d}x}\right)_e$ at cell faces using temperature values at cell centers. If thermal conductivity k and/or cross-section area A varies, they must also be evaluated. Cross-section area A is usually independent of the temperature and requires no special care. Thermal conductivity may also be known independent from temperature and directly available from the geometry. Let us now consider the more general case when thermal conductivity depends on temperature.

The most common way to calculate the thermal conductivity at node temperatures and use linear interpolation to calculate face values. For uniform mesh size linear interpolation gives

$$
\boxed{\begin{aligned} k_w &= \frac{k_W + k_P}{2} \\ k_e &= \frac{k_E + k_P}{2} \end{aligned}} \tag{2.2}
$$

see Fig. 2.4 for illustration. If we have constant thermal conductivity k we can ignore the interpolation in Eq. 2.2.

The most common way to discretize gradient term is to use central differencing. Similarly to linear interpolation, the basic idea is to assume a linear temperature profile between two nodes as shown in Fig. 2.4.

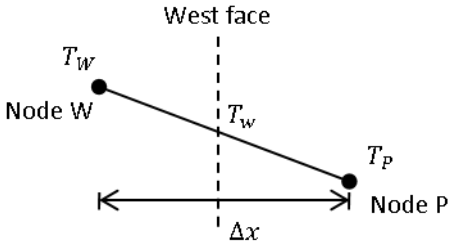


Figure 2.4: Central differencing

With uniform grid central differencing scheme gives for a temperature gradients at faces

$$
\boxed{\begin{aligned} \left(\frac{\mathrm{d}T}{\mathrm{d}x}\right)_w &\approx \frac{T_P - T_W}{\Delta x} \\ \left(\frac{\mathrm{d}T}{\mathrm{d}x}\right)_e &\approx \frac{T_E - T_P}{\Delta x} \end{aligned}} \tag{2.3}
$$

Note that for non-uniform grid where the Δx is not constant the form is slightly more complex.

Substituting Eq. 2.3 into Eq. 2.1 gives

$$k_e A_e \frac{T_E - T_P}{\Delta x} - k_w A_w \frac{T_P - T_W}{\Delta x} + \overline{S}\Delta V = 0 \tag{2.4}$$

Now we have a fully discretized the equation for heat conduction!

It is often useful to rearrange Eq. 2.4 as

$$\begin{aligned} &\frac{k_e A_e}{\Delta x} T_E - \frac{k_e A_e}{\Delta x} T_P - \frac{k_w A_w}{\Delta x} T_P + \frac{k_w A_w}{\Delta x} T_W + \overline{S}\Delta V = 0 \\ &\quad \Leftrightarrow \underbrace{\left(\frac{k_e A_e}{\Delta x} + \frac{k_w A_w}{\Delta x}\right)}_{a_P = a_W + a_E} T_P = \underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_w} T_W + \underbrace{\left(\frac{k_e A_e}{\Delta x}\right)}_{a_E} T_E + \underbrace{\overline{S}\Delta V}_{S_u} \end{aligned} \tag{2.5}$$

$$\Leftrightarrow \boxed{a_P T_P = a_W T_W + a_E T_E + S_u} \tag{2.6}$$

From Eq. 2.6 it is easy to see the affect of different terms on T_P. If west side neighbor temperature T_W increases it will increase temperature T_P and the contribution is proportional to $\frac{a_W}{a_P}$.

At this point our equation is ready for a computer, but let us make some sanity checks first.

We know that thermal conductivity k, cross-section area A and cell size Δx are positive numbers. Therefore $a = \frac{kA}{\Delta x}$ is also a positive number. Now, imagine that the temperature in the neighboring cell T_W rises. What happens to the temperature in the studied cell T_P? According to Eq. 2.6 it rises because a_W is positive. This agrees with our experience. And what if we increase thermal conductivity? Multiplier a_W increases and T_P reacts more to changes in T_W. Again, this agrees with our experience. Running a similar though experiment for a_E and S_u gives similar results and we can conclude that there is no obvious errors in our formulation.

2.1.1 Same thing in math

For one-dimensional steady heat conduction (diffusion) with a volumetric source term the governing equation is

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(k\frac{\mathrm{dT}}{dx}\right) + S = 0 \tag{2.7}$$

where T is the temperature and S is the volumetric source term. Integrating over a control volume gives

$$\int_{CV} \frac{\mathrm{d}}{\mathrm{d}x}\left(k\frac{\mathrm{dT}}{dx}\right)\mathrm{d}V + \int_{CV} S\mathrm{d}V = 0 \tag{2.8}$$

Using the Gauss divergence theorem we get

$$\begin{aligned} &\int_A \boldsymbol{n}\cdot k\frac{\mathrm{dT}}{dx}\mathrm{d}A + \int_{CV} S\mathrm{d}V = 0 \\ \Leftrightarrow &\left(kA\frac{\mathrm{dT}}{dx}\right)_e - \left(kA\frac{\mathrm{dT}}{dx}\right)_w + \overline{S}\Delta V = 0 \end{aligned} \tag{2.9}$$

With central differencing, see Eqs. 2.2 and 2.3

$$k_e A_e \frac{T_E - T_P}{\Delta x} - k_w A_w \frac{T_P - T_W}{\Delta x} + \overline{S}\Delta V = 0 \tag{2.10}$$

which is the same equations as Eq. 2.4. Rearranging Eq. 2.10 results in Eq. 2.5.

$$\underbrace{\left(\frac{k_e A_e}{\Delta x} + \frac{k_w A_w}{\Delta x}\right)}_{a_P = a_W + a_E} T_P = \underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_w} T_W + \underbrace{\left(\frac{k_e A_e}{\Delta x}\right)}_{a_E} T_E + \underbrace{\overline{S}\Delta V}_{S_u} \tag{2.11}$$

$$\Leftrightarrow \boxed{a_P T_P = a_W T_W + a_E T_E + S_u} \tag{2.12}$$

2.2 Boundary conditions

In Section 2.1 we developed the discretized equations for 1D heat transfer inside the domain. In order to use the equation 2.5 for anything real we need to give it boundary conditions.

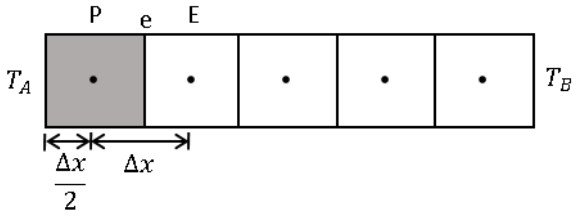


Figure 2.5: Left boundary

The cells that touch boundaries are called boundary cells and require special attention. Let us consider cell 1 in Fig. 2.5.

2.2.1 Constant temperature boundary

See Versteeg and Malalasekare [6] page 118, Example 4.1.

We define a **constant temperature T_A for left boundary**. We note that the distance from node 1 to face A is $\Delta x/2$ (not all FVM meshes are build this way but the ones we use are). Using the same interpolation and central differencing schemes as before, Eqs. 2.2 and 2.3, we get for the west side face in Eq. 2.1

$$\left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_w \approx k_A A_A \frac{T_P - T_A}{\Delta x/2} \tag{2.13}$$

Substituting this in Eq. 2.1 and treating the inside face w in the same way as before results in

$$k_e A_e \frac{T_E - T_P}{\Delta x} - k_A A_A \frac{T_P - T_A}{\Delta x/2} + \overline{S}\Delta V = 0 \tag{2.14}$$

Rearranging Eq. 2.14 gives

$$\underbrace{\left(\frac{k_e A_e}{\Delta x} + \overbrace{\frac{k_A A_A}{\Delta x/2}}^{S_P}\right)}_{a_P = a_E + S_P} T_P = \underbrace{\left(\frac{k_e A_e}{\Delta x}\right)}_{a_E} T_E + \underbrace{\left(\frac{k_A A_A}{\Delta x/2}\right) T_A + \overline{S}\Delta V}_{S_u} \tag{2.15}$$

$$\Leftrightarrow a_P T_P = a_E T_E + S_u$$

Comparing Eqs. 2.5 and 2.15 we notice a remarkable similarity. The *east* side coefficient a_E is unaffected. *West* side face doesn't exist and is replaced with source terms S_P and $\left(\frac{k_A A_A}{\Delta x/2}\right) T_A$. The source term $\overline{S}\Delta V$ stays unchanged. This is typical to boundary conditions are simplifies matrix assembly. We can first assemble the inner face coefficients and volumetric sources and worry about boundaries later. More about matrix assembly later in Sec. 2.3.

If the constant temperature is set on the B boundary as shown in Fig. 2.5 we follow a similar procedure as for A boundary. Now we replace East side coefficient a_E with suitable source terms and arrive to

$$\underbrace{\left(\frac{k_w A_w}{\Delta x} + \overbrace{\frac{k_B A_B}{\Delta x/2}}^{S_P}\right)}_{a_P = a_W + S_P} T_P = \underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_W} T_W + \underbrace{\left(\frac{k_B A_B}{\Delta x/2}\right) T_B + \overline{S}\Delta V}_{S_u} \tag{2.16}$$

$$\Leftrightarrow a_P T_P = a_W T_W + S_u$$

2.2.2 Zero gradient (insulated) boundary

See Versteeg and Malalasekare [6] page 125, Example 4.3.

Zero gradient boundary for temperature in heat conduction context means insulations as

$$q = -k \underbrace{\frac{\mathrm{d}T}{\mathrm{d}x}}_{0} = 0 \tag{2.17}$$

To apply the zero gradient boundary condition for East face in Eq. 2.1 we but the east side coefficient to zero as $a_E = 0$ and from Eq. 2.5 we get

$$\underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_P = a_W} T_P = \underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_w} T_W + \underbrace{\overline{S}\Delta V}_{S_u} \tag{2.18}$$

$$\Leftrightarrow a_P T_P = a_W T_W + S_u$$

No additional source terms are needed.

2.3 Matrix assembly

In previous section we have learned how to formulate the equations for our problem. Now we learn how to assemble these equations into a matrix form and feed it to a computer. We use Python for calculations. We want the problem in linear matrix form

$$\begin{aligned} \boldsymbol{AT} &= \boldsymbol{b} \\ \Leftrightarrow \boldsymbol{T} &= \boldsymbol{A^{-1}b} \end{aligned} \tag{2.19}$$

which can be readily solved with almost any programming language. For example in Python, using Scipy reads

```
# Solution
T = sp.linalg.solve(A,b)
```

Figure 2.6: Linear system solution with Scipy

We but all the constant values that do not depend on T in source vector $\boldsymbol{b}$ and all the multiplying coefficients of temperatures in matrix $\boldsymbol{A}$.

Let us consider one-dimensional heat conduction where the left boundary is set at constant temperature T_A and right boundary is insulated, i.e. $\frac{\mathrm{dT}}{\mathrm{dx}} = 0$. We have constant fluid properties and cross-section are A. The domain length is L and we divide it in n volumes, each $\Delta x = L/n$ long. We start the cell numbering from left and use an uniform mesh. We have uniform volumetric heat generation $\overline{S}$.

For the inside cells, dividing by the constant area A we have from Eq. 2.5

$$\begin{aligned} &\underbrace{\left(\frac{k}{\Delta x} + \frac{k}{\Delta x}\right)}_{a_P = a_W + a_E} T_P = \underbrace{\left(\frac{k}{\Delta x}\right)}_{a_w} T_W + \underbrace{\left(\frac{k}{\Delta x}\right)}_{a_E} T_E + \underbrace{S\Delta x}_{S_u} \\ &\Leftrightarrow a_P T_P = a_W T_W + a_E T_E + S_u \end{aligned} \tag{2.20}$$

For left boundary, dividing by the constant area A we have from Eq. 2.15

$$\begin{aligned} &\underbrace{\left(\frac{k}{\Delta x} + \frac{\overbrace{k}^{S_P}}{\Delta x/2}\right)}_{a_P = a_E + S_P} T_P = \underbrace{\left(\frac{k}{\Delta x}\right)}_{a_E} T_E + \underbrace{\left(\frac{k}{\Delta x/2}\right) T_A + \overline{S}\Delta V}_{S_u} \\ &\Leftrightarrow a_P T_P = a_E T_E + S_u \end{aligned} \tag{2.21}$$

For right boundary, dividing by the constant area A we have from Eq. 2.18

$$\begin{aligned} &\underbrace{\left(\frac{k}{\Delta x}\right)}_{a_P = a_W} T_P = \underbrace{\left(\frac{k}{\Delta x}\right)}_{a_w} T_W + \underbrace{\overline{S}\Delta V}_{S_u} \\ &\Leftrightarrow a_P T_P = a_W T_W + S_u \end{aligned} \tag{2.22}$$

We may now observe from Eqs. 2.20-2.22 that the for all cells that do have a west boundary, the west face coefficient is the same $a_W = \frac{k}{\Delta x}$. The west face coefficient operates on the currently studied cell T_P and the west side neighbor T_W.

Adding the west side coefficients into matrix $\boldsymbol{A}$ gives

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -a_W & a_W & 0 & 0 & 0 \\ 0 & -a_W & a_W & 0 & 0 \\ 0 & 0 & -a_W & a_W & 0 \\ 0 & 0 & 0 & -a_W & a_W \end{bmatrix} \tag{2.23}$$

With python this can be done as

```
# Coefficient matrix
A = sp.zeros((n,n))

# Add aW to matrix A
for k in range(1,n):
    A[k,k]   += aW
    A[k,k-1] += -aW
```

Similarly, for all east faces that exist the east face coefficient $a_E = \frac{k}{\Delta x}$ and the coefficient operates on the current cell and the east neighbor. Adding the east coefficients to already build matrix in Eq. 2.23 gives

$$\boldsymbol{A} = \begin{bmatrix} a_E & -a_E & 0 & 0 & 0 \\ -a_W & a_W + a_E & -a_E & 0 & 0 \\ 0 & -a_W & a_W + a_E & -a_E & 0 \\ 0 & 0 & -a_W & a_W + a_E & -a_E \\ 0 & 0 & 0 & -a_W & a_W \end{bmatrix} \tag{2.24}$$

With python

```
# Add aE to matrix A
for k in range(n-1):
    A[k,k]   += aE
    A[k,k+1] += -aE
```

Now we need to add the boundary conditions to our matrix $\boldsymbol{A}$ and source vector $\boldsymbol{b}$.

As can be seen from Eq. 2.21 the constant temperature boundary condition on the left contributes terms $S_P = \frac{k}{\Delta x/2}$ to the matrix $\boldsymbol{A}$ first cell diagonal. Adding that term gives

$$\boldsymbol{A} = \begin{bmatrix} a_E + \frac{k}{\Delta x/2} & -a_E & 0 & 0 & 0 \\ -a_W & a_W + a_E & -a_E & 0 & 0 \\ 0 & -a_W & a_W + a_E & -a_E & 0 \\ 0 & 0 & -a_W & a_W + a_E & -a_E \\ 0 & 0 & 0 & -a_W & a_W \end{bmatrix} \tag{2.25}$$

The constant boundary condition also contributes a source term $\left(\frac{k}{\Delta x/2}\right) T_A$ to the first cell of the source vector. The resulting source vector

$$\boldsymbol{b} = \begin{bmatrix} \left(\frac{k}{\Delta x/2}\right) T_B & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{2.26}$$

With python the constant boundary condition is added as

```
# Source vector
b = sp.zeros(n)

# Boundary  on the left
A[0,0] += k/(dx/2)
b[0]   += k/(dx/2)*T_B
```

The insulated boundary on the right produces no coefficients as no heat is transfered through the boundary. Nothing needs to be done.

As the final step we need to add the volumetric source to our source vector $\boldsymbol{b}$. The volumetric source is the same in all Eqs. 2.20-2.22, $S_u = S\Delta x$. By adding it to the source vector we get

$$\boldsymbol{b} = \begin{bmatrix} \left(\frac{k}{\Delta x/2}\right) T_B + S\Delta x & S\Delta x & S\Delta x & S\Delta x & S\Delta x \end{bmatrix}^T \tag{2.27}$$

With python this can be done with vectorized statement as

```
# Add heat generation
b += S*dx
```

Now our linear system is ready and we can solve as shown in Fig. 2.6. The full system reads

$$\underbrace{\begin{bmatrix} a_E + \frac{k}{\Delta x/2} & -a_E & 0 & 0 & 0 \\ -a_W & a_W + a_E & -a_E & 0 & 0 \\ 0 & -a_W & a_W + a_E & -a_E & 0 \\ 0 & 0 & -a_W & a_W + a_E & -a_E \\ 0 & 0 & 0 & -a_W & a_W \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix}}_{\boldsymbol{T}} = \underbrace{\begin{bmatrix} \left(\frac{k}{\Delta x/2}\right) T_B + S\Delta x \\ S\Delta x \\ S\Delta x \\ S\Delta x \\ S\Delta x \end{bmatrix}}_{\boldsymbol{b}}$$

It's a good idea to check your code in as small increments as possible. Print out your coefficients and matrices during development and compare them to known values. Debugging of small pieces is easier that large ones. Software houses have teams of designated testers working in their companies with the sole purpose of finding bugs. Do it early.

You can plot the solutions with python as

```
# Plot numerical and exact solution
x = sp.linspace(dx/2,L-dx/2,n)
plt.plot(x, T, "k--o", label="numerical")
plt.legend()
```

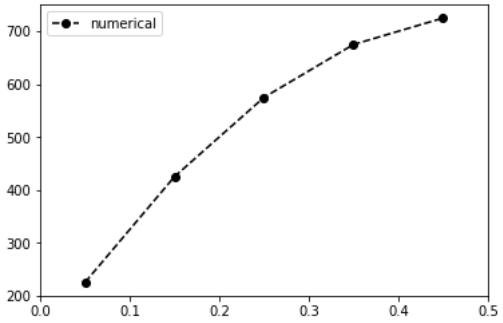


Figure 2.7: Plotting

2.4 Unsteady

See Versteeg and Malalasekare [6] page 243, Chapter 8.

There are many different methods to tackle the time derivative term in one-dimensional heat conduction

$$\rho c \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right) + S \tag{2.28}$$

were ρ is density, c is specific heat capacity. The partial derivate $\frac{\partial T}{\partial t}$ mean that we now have two different derivatives, $\frac{\partial T}{\partial t}$ and $\frac{\partial T}{\partial x}$.

We will try to avoid mathematics as much as possible. For a more formal approach see *Versteeg and Malalasekare [6].*

Let us first consider the left hand side of Eq. 2.28, $\rho c \frac{\partial T}{\partial t}$. We may discretized it as

$$\rho c \frac{\partial T}{\partial t} \approx \rho c \frac{T - T^0}{\Delta t} \tag{2.29}$$

where Δt is time step and T^0 is temperature from previous time step. If we now substitute Eq. 2.29 into Eq. 2.28 we get

$$\rho c \frac{T - T^0}{\Delta t} = \frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right) + S \tag{2.30}$$

Using FVM to the left hand side of Eq. 2.30

$$\int_{CV} \rho c \frac{T - T^0}{\Delta t} dV \approx \rho c A \frac{T - T^0}{\Delta t} \Delta x \tag{2.31}$$

We may now recognize the right hand side of Eq. 2.31, $\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + S$ as the governing equation for steady state heat conduction, Eq. 2.7. We may use the FVM methods described before to discretize it.

The remaining choice is at what time to we evaluate the right hand side of Eq. 2.30. If we choose to evaluate it at the last time step corresponding to T^0 we get explicit Euler scheme. Explicit Euler requires very small time steps and is rarely suitable.

We choose to use the new time. This is called implicit Euler scheme. Implicit Euler is unconditionally stable and easy to use. For more temporal accuracy look in to Runge-Kutta or other higher order schemes.

With FVM and Implicit Euler we get from Eqs. 2.30, 2.31, 2.5.

$$\begin{aligned}
&\underbrace{\rho c A \frac{\Delta x}{\Delta t}}_{a_P^0}(T_P - T_P^0) + \underbrace{\left(\frac{k_e A_e}{\Delta x} + \frac{k_w A_w}{\Delta x}\right)}_{a_W + a_E} T_P = \underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_w} T_W + \underbrace{\left(\frac{k_e A_e}{\Delta x}\right)}_{a_E} T_E + \underbrace{\overline{S}\Delta V}_{S_u} \\
&\Leftrightarrow a_P^0 T_P - a_P^0 T_P^0 + a_W T_P + a_E T_P = a_w T_W + a_E T_E + S_u \\
&\Leftrightarrow \boxed{\underbrace{(a_P^0 + a_W + a_E)}_{a_P} T_P = a_w T_W + a_E T_E + a_P^0 T_P^0 + S_u}
\end{aligned} \tag{2.32}$$

From Eq. 2.32 we see that the solution of one time step in unsteady case is very similar to the steady one. We only have two extra terms, a_P^0 in the a_P term and $a_P^0 T_P^0$. The a_P^0 remains constant during time steps and can be added to the coefficient matrix. In Python

```
# Add time coefficient to diagonal
for k in range(n):
    A[k,k] += aP0
```

The source term $a_P^0 T_P^0$ changes with temperature and has to updated during every time step. It is often convenient to collect the time-independent terms into a separate source vector, see *bConstant* in Fig. 2.8. This often results in simpler and faster code.

To progress the solution repeat the process in a loop. An example is given in Fig. 2.8.

```
# Solution
for step in range(1,steps+1):
    b  = bConstant + aP0*T
    T  = sp.linalg.solve(A,b)
    Ts[step] = T
    t += dt
```

Figure 2.8: Time stepping

2.5 2D and 3D cases

See Versteeg and Malalasekare [6] page 129, Chapter 4.4-4.5.

Extension from 1D to 2D or 3D is straight forward. We again form a control volume and add all the fluxes in a balance equation. It is customary to use North and South (N,S) as names for the upper and lower neighbors. See Fig. 2.9.

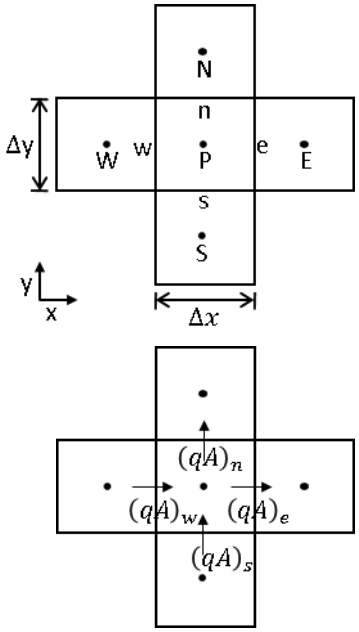


Figure 2.9: 2D notations and control volume

Forming a balance equation for 2D heat conduction results in

$$\begin{aligned} (qA)_w - (qA)_e + (qA)_s - (qA)_n + \overline{S}\Delta V = 0 \\ \Leftrightarrow \left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_e - \left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_w + \left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_n - \left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_s + \overline{S}\Delta V = 0 \end{aligned} \quad (2.33)$$

With finite volume method, using central differencing and linear interpolation Eq. 2.33 gives

$$\begin{aligned} \underbrace{\left(\frac{k_eA_e}{\Delta x} + \frac{k_wA_w}{\Delta x} + \frac{k_sA_s}{\Delta y} + \frac{k_nA_n}{\Delta y}\right)}_{a_P = a_W + a_E + a_S + a_N} T_P = \underbrace{\left(\frac{k_wA_w}{\Delta x}\right)}_{a_W} T_W + \underbrace{\left(\frac{k_eA_e}{\Delta x}\right)}_{a_E} T_E \\ + \underbrace{\left(\frac{k_sA_s}{\Delta y}\right)}_{a_S} T_S + \underbrace{\left(\frac{k_nA_n}{\Delta y}\right)}_{a_N} T_N \\ + \underbrace{\overline{S}\Delta V}_{S_u} \end{aligned} \quad (2.34)$$

$$\Leftrightarrow \boxed{a_P T_P = a_W T_W + a_E T_E + a_S T_S + a_N T_N + S_u} \tag{2.35}$$

As we can see from Eqs. 2.34-2.35 the extension from 1D to 2D only bring a couple of new terms to the equation. The structure remains the same.

Similarly, extension from 2D to 3D would give two more terms. The extra directions are usually named Bottom and Top (B and T).

The author would recommend motivated students to derive the Eq. 2.34 and to program a 2D solver. The geometry and code is still relatively easy to understand. 3D solver code becomes rather complicated and mostly teaches programming with very little new to learn about FVM.

If you program your own solver in 2D or 3D, it is highly recommended to use sparse matrices, see extra material in Section 2.7.1.

2.6 Notations

One we move from 1D to 2D or 3D, the full notation of partial differential equations quickly becomes cumbersome. Using a 3D heat conduction as an example the full form is

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right) + \frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right) + S = 0 \tag{2.36}$$

it is customary to use either a vector or tensor notation instead. With vector notation Eq. 2.36 becomes

$$\nabla \cdot k\nabla T + S = 0 \tag{2.37}$$

and with tensor notation

$$\frac{\partial}{\partial x_j}\left(k\frac{\partial T}{\partial x_j}\right) + S = 0 \tag{2.38}$$

Both vector and tensor notations are widely used in the literature. It's rather necessary to understand them and they will be used on this course also. For index notation see Appendix A.1.

2.7 Extra material

Extra material. Not needed on this course.

2.7.1 Performance issues with example codes

Extra material. Not needed on this course.

All the examples are written with clarity in mind. They usually sacrifice performance for this goal. The most important performance limiting factor is the use of dense matrices.

The multiplying matrix $\boldsymbol{A}$ is mostly empty in most FVM problems, i.e. mostly filled with zeros. It would be a LOT faster to use sparse matrices. With large systems sparse matrices are decades faster that dense matrices. Sparseness also saves memory.

The Python syntax for sparse matrices is rather verbose, but straight forward to use. Do not be frighten by the long commands. You can usually just copy-paste it from your previous codes or create a wrapper. For an example wrapper see Fig. 2.10. For more detail see https://docs.scipy.org/doc/scipy/reference/sparse.html.

```
11 import scipy.sparse as sparse
12 from scipy.sparse import linalg as splinalg
13
14 class SparseMatrixA(object):
15     def __init__(self):
16         self.row = []
17         self.col = []
18         self.val = []
19
20     def add(self, row, col, val):
21         self.row.append(row)
22         self.col.append(col)
23         self.val.append(val)
24
25     def finalize(self):
26         return sparse.csr_matrix(sparse.coo_matrix(
27                                  (self.val, [self.row, self.col])
28                                 ))
29
30     def solve(self,b, A=None):
31         if A is None:
32             A = self.finalize()
33         return splinalg.spsolve(A, b)
```

Figure 2.10: Sparse matrix wrapper class

We often use unnecessary loops. Looping is slow in Python and indexing of large matrices is also unnecessary. As rule of a thumb, always use vectorized commands if you can.

2.7.2 Application to other physics

Extra material. Not needed on this course.

Many problems in other fields are very similar to heat conduction. Here are some examples.

Electric heating Electric potential be solved from the well-known Poisson equation (same as heat conduction equation)

$$\nabla \cdot \sigma \nabla \phi = 0 \tag{2.39}$$

where ϕ is electric potential. Electric field $\boldsymbol{E}$ is solved from

$$\boldsymbol{E} = -\nabla \phi \tag{2.40}$$

and electric current density $\boldsymbol{J}$ from

$$\boldsymbol{J} = \sigma \boldsymbol{E} \tag{2.41}$$

and finally the volumetric electric heating power p

$$p = \frac{\partial P}{\partial V} = \boldsymbol{J} \cdot \boldsymbol{E} = \boldsymbol{J} \cdot \boldsymbol{J} / \boldsymbol{\sigma} = \frac{\mid \boldsymbol{J} \mid^{\mathbf{2}}}{\boldsymbol{\sigma}} \tag{2.42}$$

Electric conductivity can be calculated from electric resistivity ρ as

$$\sigma = \frac{1}{\rho} \tag{2.43}$$

Linear Mechanics For small deformations

$$-\nabla \cdot \boldsymbol{\sigma} = \boldsymbol{f} \tag{2.44}$$

$$\boldsymbol{\sigma} = \lambda \nabla \cdot \boldsymbol{u} \mathbf{I} + 2\mu \boldsymbol{D} \tag{2.45}$$

The resulting equation is similar linear system as for heat conduction and can be solved with same methods.

The stress equation is very similar as that in Navier-Stokes equations. If we remove movement related terms from Navier-Stokes equations, we end up with an equation very similar to Eq. 2.44.

3 Heat advection and conduction

The topics discussed in this section may be found in a from Versteeg and Malalasekara 2007, Chapter 5. [6]

In this section we learn to extend our knowledge of finite volume method from pure heat conduction to combined heat advection and conduction. Depending on the source, advection if often called convection.

The equations in this section are written in terms of heat. The same methods apply to any other advection diffusion problem.

In this section we assume that the velocities at cell faces are known. We will leave the solution methods for Navier Stokes equations for later section. However, the methods discussed in this section apply to Navier Stokes solution and after studying this section you will find many familiar key words from CFD software.

3.1 Governing equation

In this section, we will learn to develop equations for one dimensional heat advection conduction problem with internal heat generation.

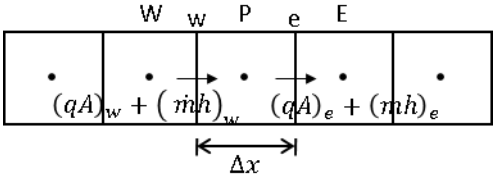


Figure 3.1: Heat advection and convection

Similarly to the heat conduction problem earlier, we start with energy balance equation for the control volume P in Fig. 3.1

$$(qA)_w - (qA)_e + (\dot{m}h)_w - (\dot{m}h)_e + \overline{S}\Delta V = 0 \tag{3.1}$$

where mass flow rate is $\dot{m} = \rho u A$ and specific enthalpy $h = cT$. Velocity is u, specific heat capasity is c. Using the methods described in the previous section for conduction and expanding the advection terms we get

$$k_e A_e \frac{T_E - T_P}{\Delta x} - k_w A_w \frac{T_P - T_W}{\Delta x} + \rho_w u_w A_w c_w T_w - \rho_e u_e A_e c_e T_e + \overline{S}\Delta V = 0 \tag{3.2}$$

Continuity is worth mentioning here, as it is closely linked to advection. From continuity we get

$$\begin{aligned} \dot{m}_w - \dot{m}_e &= 0 \\ \Leftrightarrow (\rho u A)_w - (\rho u A)_e &= 0 \end{aligned} \tag{3.3}$$

which states that mass is conserved. Let us now assume that density, velocity, and specific heat is somehow known at the cell faces. This leaves the face temperatures T_w and T_e as the only variables requiring further study.

The advection term discretization scheme is one of the choices that must be done almost every time computational fluid dynamics is used. It has a large effect on both the accuracy and stability of the model. Great many methods exists in the literature and in the software.

We will study two methods in detail, the linear interpolation and upwind scheme. Most of commonly used methods are some kind of combinations of these two main ideas, or at least very similar. We will also shortly explain the main idea of linear upwind scheme, which is one of the most popular advection schemes.

3.2 Linear interpolation

We have already discussed linear interpolation in the context of heat conduction, see Eq. 2.2. Linear interpolation is often called central differencing scheme (CDS) in context of computational fluid dynamics.

For uniform grid we get for face temperatures

$$\begin{aligned} T_w &= \frac{T_W + T_P}{2} \\ T_e &= \frac{T_E + T_P}{2} \end{aligned} \tag{3.4}$$

Using Eqs. 3.2 and 3.4 we get for the inner cells

$$\begin{aligned} k_e A_e \tfrac{T_E - T_P}{\Delta x} - k_w A_w \tfrac{T_P - T_W}{\Delta x} + \rho_w u_w A_w c_w \tfrac{T_W + T_P}{2} \\ -\rho_e u_e A_e c_e \tfrac{T_E + T_P}{2} + \overline{S}\Delta V = 0 \end{aligned} \tag{3.5}$$

$$\begin{aligned} \Leftrightarrow \left(\underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} + \underbrace{\frac{k_w A_w}{\Delta x}}_{a_{Wc}} - \underbrace{\frac{\rho_w u_w A_w c_w}{2}}_{a_{Wf}} + \underbrace{\frac{\rho_e u_e A_e c_e}{2}}_{a_{Ef}} \right) T_P \\ = \left(\underbrace{\frac{k_w A_w}{\Delta x}}_{a_{Wc}} + \underbrace{\frac{\rho_w u_w A_w c_w}{2}}_{a_{Wf}} \right) T_W + \left(\underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} - \underbrace{\frac{\rho_e u_e A_e c_e}{2}}_{a_{Ef}} \right) T_E + \underbrace{\overline{S}\Delta V}_{S_u} \end{aligned} \tag{3.6}$$

where we have used subscripts c and f to refer to conduction and advection (flow), respectively. If velocity u is positive, all coefficients are positive. If velocity is negative, coefficients a_{Wf} and a_{Ef} are negative. Using the short hand notation for matrix coefficients we get

$$(a_{Ec} + a_{Wc} - a_{Wf} + a_{Ef})\, T_P = (a_{Wc} + a_{Wf})\, T_W + (a_{Ec} - a_{Ef})\, T_E + S_u \tag{3.7}$$

which is ready for a computer solution. Note the negative terms $-a_{Wf}$ and $-a_{Ef}$. Negative terms risk unboundedness, see Section 3.6.3.

Even though linear interpolation for advection is second order accurate, see Sec. 3.6.1, it is rarely used in computational fluid dynamics. The two main flaws

are the negative coefficients that allow a numerical solution to become unstable and that the nature of advection is not well presented in linear interpolation. Advection moves information from upwind to down wind. The advection term that is discretized with linear interpolation moves information in all directions.

3.3 Upwind scheme

Upwind scheme is the simplest upwind biased scheme. In upwind scheme we assume that the face value is the same as the node value on the upwind side. For positive velocity this gives

$$\begin{aligned} T_w &= T_W \\ T_e &= T_P \end{aligned} \tag{3.8}$$

and for negative velocity

$$\begin{aligned} T_w &= T_P \\ T_e &= T_E \end{aligned} \tag{3.9}$$

Using the upwind scheme with **positive velocity** and energy balance, Eqs. 3.2 and 3.8, we get for the inner cells

$$\begin{aligned} k_e A_e \tfrac{T_E - T_P}{\Delta x} - k_w A_w \tfrac{T_P - T_W}{\Delta x} + \rho_w u_w A_w c_w T_W \\ -\rho_e u_e A_e c_e T_P + \overline{S}\Delta V = 0 \end{aligned} \tag{3.10}$$

$$\begin{aligned} \Leftrightarrow & \left(\underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} + \underbrace{\frac{k_w A_w}{\Delta x}}_{a_{Wc}} + \underbrace{\rho_e u_e A_e c_e}_{a_{Ef}} \right) T_P \\ & = \left(\underbrace{\frac{k_w A_w}{\Delta x}}_{a_{Wc}} + \underbrace{\rho_w u_w A_w c_w}_{a_{Wf}} \right) T_W + \left(\underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} \right) T_E + \underbrace{\overline{S}\Delta V}_{S_u} \end{aligned} \tag{3.11}$$

For **negative velocity**, using Eqs. 3.2 and 3.9, we similarly get

$$\begin{aligned} k_e A_e \tfrac{T_E - T_P}{\Delta x} - k_w A_w \tfrac{T_P - T_W}{\Delta x} + \rho_w u_w A_w c_w T_P \\ -\rho_e u_e A_e c_e T_E + \overline{S}\Delta V = 0 \end{aligned} \tag{3.12}$$

$$\begin{aligned} \Leftrightarrow & \left(\underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} + \underbrace{\frac{k_w A_w}{\Delta x}}_{a_{Wc}} - \underbrace{\rho_w u_w A_w c_w}_{a_{Wf}} \right) T_P \\ & = \left(\underbrace{\frac{k_w A_w}{\Delta x}}_{a_{Wc}} \right) T_W + \left(\underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} - \underbrace{\rho_e u_e A_e c_e}_{a_{Ef}} \right) T_E + \underbrace{\overline{S}\Delta V}_{S_u} \end{aligned} \tag{3.13}$$

With suitable max/min operators it possible to write the positive and negative velocity cases in the same equation.

Keeping in mind the sign of the velocity, all the coefficients are unconditionally positive. Together with some other mathematical properties, this makes upwind scheme very stable. The scheme is also upwind biased and represents the advection phenomena well. However, because of the very simple assumptions used in upwind scheme, the scheme is only first order accurate, see Sec. 3.6.1.

Because of the stability of the upwind scheme it is widely used in the early stages of a simulation. Once the simulation has produced a somewhat physical solution, it's easy to change the scheme to a more accurate one.

3.4 Linear upwind

Linear upwind is one of the preferred schemes for advection. It is both second order accurate and upwind biased. Studying it in detail is, however, outside the scope of this course and we will only discuss the main idea here.

With linear upwind scheme, we take the upwind node value and correct it with a linear correction. For positive velocity, on a uniform mesh this gives for the west face

$$T_w = T_W + \frac{\mathrm{d}T}{\mathrm{d}x}\frac{\Delta x}{2} \tag{3.14}$$

There are many ways to estimate the gradient of temperature $\frac{\mathrm{d}T}{\mathrm{d}x}$.

Linear upwind scheme is not as numerically stable as upwind scheme. Without additional limiters it tends to produce over and undershoots.

Because of the good accuracy and stability issues there are great many variations of the method. Most aim to improve the numerical properties of the scheme, while keeping as much second order accuracy as possible. Remember to check the details when using software packages! Small differences in advection term discretization may be the difference between converging and diverging simulations.

3.5 Boundary conditions

The boundary conditions must be discretized in a consistent manner with the inner cells. Here we study constant value boundary conditions for linear interpolation and upwind scheme.

3.5.1 Constant boundary condition for linear interpolation

We will only consider a constant left boundary condition here. A constant right boundary is similar. We use subscript A for the left boundary.

Starting from energy balance we get

$$(qA)_A - (qA)_e + (\rho u A c T)_A - (\rho u A c T)_e + \overline{S}\Delta V = 0 \tag{3.15}$$

Conduction terms are handled as explained in previous sections. We use central differencing for the east face advection. There is no need to do anything for the A face term, as it is known. We get

$$k_e A_e \frac{T_E - T_P}{\Delta x} - k_A A_A \frac{T_P - T_A}{\Delta x/2} + \rho_A u_A A_A c_A T_A - \rho_e u_e A_e c_e \frac{T_E + T_P}{2} + \overline{S}\Delta V = 0 \tag{3.16}$$

Rearranging gives

$$\begin{aligned} &\left(\underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} + \overbrace{\frac{k_A A_A}{\Delta x/2}}^{S_P} + \underbrace{\frac{\rho_e u_e A_e c_e}{2}}_{a_{Ef}} \right) T_P \\ &\quad = \left(\underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} - \underbrace{\frac{\rho_e u_e A_e c_e}{2}}_{a_{Ef}} \right) T_E + \underbrace{\left(\frac{k_A A_A}{\Delta x/2} \right) T_A + \rho_A u_A A_A c_A T_A + \overline{S}\Delta V}_{S_u} \end{aligned} \tag{3.17}$$

3.5.2 Constant boundary condition for upwind scheme

We will only consider **positive velocities** here. Negative velocities are considered similarly. We use subscript A for the left boundary and subscript B for the right boundary.

Starting from energy balance we get for the **left boundary**

$$(qA)_A - (qA)_e + (\rho u A c T)_A - (\rho u A c T)_e + \overline{S}\Delta V = 0 \tag{3.18}$$

Using central differencing for conduction terms and upwind scheme for advection terms gives

$$\left(\underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} + \overbrace{\frac{k_A A_A}{\Delta x/2}}^{S_P} + \underbrace{\rho_e u_e A_e c_e}_{a_{Ef}} \right) T_P = \left(\underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} \right) T_E + \underbrace{\left(\frac{k_A A_A}{\Delta x/2} \right) T_A + \rho_A u_A A_A c_A T_A + \overline{S}\Delta V}_{S_u} \tag{3.19}$$

The known face A value can be used directly for the face A. Similarly, for the **right boundary**

$$(qA)_w - (qA)_B + (\rho u A c T)_w - (\rho u A c T)_B + \overline{S}\Delta V = 0 \tag{3.20}$$

$$\left(\underbrace{\frac{k_w A_w}{\Delta x}}_{a_{Wc}} + \overbrace{\frac{k_B A_B}{\Delta x/2}}^{S_P} + \rho_B u_B A_B c_B \right) T_P = \left(\underbrace{\frac{k_w A_w}{\Delta x}}_{a_{wc}} + \underbrace{\rho_w u_w A_w c_w}_{a_{Wf}} \right) T_W + \underbrace{\left(\frac{k_B A_B}{\Delta x/2} \right) T_B + \overline{S}\Delta V}_{S_u} \tag{3.21}$$

Note that the known value T_B **is not used for advection at the outlet face**!

For negative velocity, a similar procedure applies.

3.6 Numerical properties of advection discretization schemes

We want our numerical schemes to be stable and accurate. Here are some criteria on how the estimate these properties.

3.6.1 Accuracy

There is no way to know how accurate a numerical solution is, unless you can compare it to some known solution. However, there are simple methods on how to estimate how fast the numerical solution will become better with a finer mesh. Using the Taylor series expansion about P gives with uniform mesh

$$T_e = T_P + \left(\frac{\mathrm{dT}}{\Delta x}\right)_P \frac{\Delta x}{2} + \left(\frac{\mathrm{d^2T}}{\Delta x^2}\right)_P \left(\frac{\Delta x}{2}\right)^2 + ... \tag{3.22}$$

Comparing Eq. 3.8 for upwind scheme, $T_e = T_P$ and Eq. 3.22 we see that upwind scheme is first order accurate. This means that the error will decrease linearly with decreasing cell size. Note, however, that such mathematical properties should not be taken too literally. Even if the error usually decreases linearly, it does not always do so. This behavior also only occurs with fine enough mesh.

Similarly, comparing Eq. 3.14 for linear upwind scheme, $T_w = T_W + \frac{\mathrm{d}T}{\mathrm{d}x}\frac{\Delta x}{2}$ and Eq. 3.22 we see that linear upwind scheme is second order accurate. In other words, halving the cell size should reduce the error to one fourth.

For linear interpolation, or central differencing, we similarly get second order accuracy.

3.6.2 Conservativeness

One of the main advantages of finite volume method in fluid dynamics is that it is conservative by nature. Conservativeness means that a flux leaving one cell through a face must be equal to the flux entering the adjacent cell. As the cell faces are linked together, the same applies for the whole domain if there are no sources. This is achieved by consistent representation of the flux in both cells. Consistent representation means using the same discretization.

Using a one dimensional source less heat conduction with three cells and constant boundary fluxes, see Fig. 3.2, as an example.

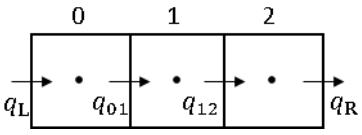


Figure 3.2: Conservativeness

Adding the terms gives

$$(q_L - q_{01}) + (q_{01} - q_{12}) + (q_{12} - q_R) = q_L - q_R \tag{3.23}$$

as it should. Using central differencing with and assuming constant heat conductivity gives

$$\left(q_L + k\frac{T_1 - T_0}{\Delta x}\right) + \left(-k\frac{T_1 - T_0}{\Delta x} + k\frac{T_2 - T_1}{\Delta x}\right) + \left(-k\frac{T_2 - T_1}{\Delta x} - q_R\right) = q_L - q_R \tag{3.24}$$

All the terms rising from the central differencing cancel out and we are left with the boundary fluxes. In other words, central differencing is conservative for heat conduction.

Similar considerations would show us that all the schemes studied here are conservative.

3.6.3 Boundedness

Boundedness mean that in the absence of sources the internal field values should be bounded by boundary values. If, for example, temperature on one boundary is one and zero on the other, all temperatures should be between one and zero. Many advection schemes lack this property.

One essential requirement for boundedness is that all the coefficients in a discretized system should have the same sign. Physically this means that if temperature rises in one cell, temperature should also rise in the neighboring cell.

Diagonal dominance is a key factor for iterative solution of a equation system. Even linear systems are usually solved iteratively. Diagonal dominance is defined as

$$\frac{\sum |a_{nb}|}{|a_P|} \begin{cases} \leq & 1 \text{ at all cells} \\ < & 1 \text{ at least in one cell} \end{cases} \tag{3.25}$$

In practice this means that we want big values at the diagonal. This becomes relevant when using nonlinear source terms like radiation. Non-linear source terms are outside the scope of this lecture note, but might be considered later on the course.

Linear interpolation and linear upwind schemes are not bounded. They might produce large over- and undershoots. There are many methods to limit this behavior for linear upwind schemes but they often degenerate it's accuracy. Upwind scheme is bounded.

Now that we know that all the coefficients of discretized equation should have the same sign, let us have a closer look at the linear interpolation scheme. From Eq. 3.6 we see that

$$a_E = \underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} - \underbrace{\frac{\rho_e u_e A_e c_e}{2}}_{a_{Ef}} \tag{3.26}$$

because

$$a_W = \underbrace{\frac{k_w A_w}{\Delta x}}_{a_{Wc}} + \underbrace{\frac{\rho_w u_w A_w c_w}{2}}_{a_{Wf}} \tag{3.27}$$

is unconditionally positive for positive value of velocity u we want to have a positive a_E as well. In other words

$$
\begin{aligned}
&\underbrace{\frac{k_e A_e}{\Delta x}}_{a_{Ec}} - \underbrace{\frac{\rho_e u_e A_e c_e}{2}}_{a_{Ef}} > 0 \\
&\Leftrightarrow 1 - \frac{\rho_e u_e c_e}{2k_e/\Delta x} > 0 \\
&\Leftrightarrow -\frac{\rho_e u_e c_e}{2k_e/\Delta x} > -1 \\
&\Leftrightarrow \mathrm{Pe} = \frac{\rho_e u_e c_e}{k_e/\Delta x} < 2
\end{aligned}
\tag{3.28}
$$

Now we have boundedness criteria for Peclet number $\mathrm{Pe} < 2$ for central differencing scheme. If $\mathrm{Pe} > 2$, a high temperature in one cell might decrease the temperature of its neighbor. This is obviously wrong.

3.6.4 Transportiveness

The discretization scheme should be able to describe the direction of the flow. Upwind biased schemes do this but linear interpolation does not.

3.6.5 Courant number

Courant number has many different names but in computational fluid dynamics context it's most often called Courant number. Courant–Friedrichs–Lewy (CFL) condition is also a often used name. Courant number is defined as

$$
\mathrm{Co} = \frac{u\Delta t}{\Delta x} < 1 \tag{3.29}
$$

and should always be smaller than unity. Imagine a infinitesimally small particle of fluid in the flow. If we want to have some physically meaningful discretization of the path of that particle, we can't allow it to "skip" cells. The particle can be allowed to move at maximum distance of one cell size during a time step. Courant number is one of the most important parameters for transient simulation stability.

The Courant number gives the maximum allowed time step. It dos not, however, guarantee a accurate simulation. It is possible that a much smaller time step is needed for other reasons.

It is sometimes possible to violate Courant number limitations if we are not interest in the transient phenomena and the case is stable enough. This is especially useful when using transient solvers to gain a steady state solution.

When solving for more complicated physics, there are often other limitations to time step that are similar to Courant number. If, for example, we are solving for super sonic flow with shock waves, it is usually needed to use a similar limitation for shock wave propagation speed (sonic speed). Limitations like this may be force us to use very small time steps or additional stabilizers.

3.7 False diffusion

The basic idea of using a upwind scheme is to use the node value from upwind side to as the face value. The value might be corrected somehow but but the basic idea is the same in all upwind schemes. If the velocity vector and face

normal vector are perfectly aligned, it is clear what the "upwind" node means. When they do not align, it is not so clear. See Fig. 3.3, where is the upwind node for north face? How about east face? For both questions the best available answer is node P.

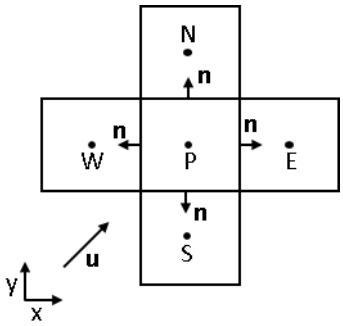


Figure 3.3: Non aligning face normals and velocity vector

Because node P is now the upwind node for two different faces, it will also contribute to the advection terms for both faces

$$\begin{aligned} a_{Nf} &= \rho_n A_n c_n \boldsymbol{n} \cdot \boldsymbol{u} \\ a_{Ef} &= \rho_e A_e c_e \boldsymbol{n} \cdot \boldsymbol{u} \end{aligned} \tag{3.30}$$

where $\boldsymbol{n}$ is the surface normal vector (different vector for different faces). This causes the heat to spread, by advection, into directions where the advection velocity is not moving. This phenomena is very similar to diffusion, and is often called false diffusion or numerical diffusion.

For the previous reason it is important that the mesh face normals align with the flow as much as possible.

A common example of an obvious false diffusion is a rectangular mesh with constant boundary conditions as shown in Fig. 3.4. We have pure advection, i.e. $k = 0\,\mathrm{W/mK}$ and the velocity is in 45°angle to the face normals.

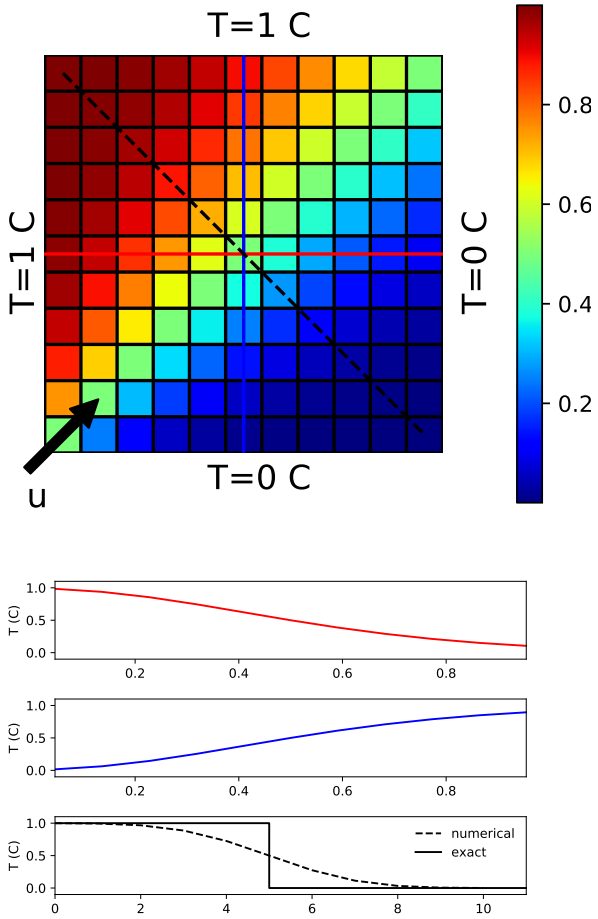


Figure 3.4: False diffusion

It is clear that an exact solution would just move the temperatures from left and bottom boundaries inside the domain. At the diagonal there would be a sharp boundary and all values would be either ones or zeros.

The false diffusion looks very similar to real diffusion and we do not get a sharp boundary from the numerical solution. This is best visualized in the bottom graph in Fig. 3.4 for the diagonal values. If we would to rotate the mesh by 45° to align with the velocity, we would get an sharp boundary and a near perfect solution to this simple problem.

False diffusion is closely related to meshing. If we are able to arrange the rectangular cells used in Fig. 3.4 so that their face normal align to flow, there is no numerical diffusion. If, on the other hand, the face normal have a 45° angle with flow, the solution is as bad as possible.

Using triangular cells, it is not possible to perfectly align the faces with the flow. But on the other hand, it is not possible to align the faces as badly as in the worst case scenario with rectangular cells. The more "chaotic" manner by

which the triangular cell faces are aligned, makes both very good alignment and very bad alignment impossible.

As a rule of the thump, it is a good idea to use rectangular cells if you are able to align the faces with the flow. If you are not able to do so, it's better to use something else. Polyhedra cells are gaining popularity over tetrahedron at the moment.

3.8 Transient

The procedure of adding transient terms to advection conduction problem is exactly the same as for pure conduction problem. We will not repeat the process here.

When solving for transient case, one must remember the Courant number constraint, Eq. 3.29, for stability. Also, Courant number only defines the largest possible time step for stability. For accuracy we may need a much smaller time step.

3.9 2D and 3D cases

Similarly to pure heat conduction considered in Section 2.5, more dimensions simply add more terms to the heat balance equation. The methods remain the same.

For the motivated it is recommended to derive the equations and program a solver by yourself. It might take a while, but afterwards you can be confident that you understand the material discussed this far. With this skill level, you will be able to solve many industrial problems. Even if such problems are easily solved with existing software packages, it is sometimes beneficial to write a custom code that does exactly what you want. This allows you to simplify the problem and use much more efficient solution methods.

4 Fluid flow

Some of the topics discussed in this section may be found from Versteeg and Malalasekara 2007, Chapter 6. [6]

In this section we learn the basics of solution methods for practical fluid flow problems. In previous section we studied diffusion and advection problems in fine enough detail to implement our own solvers. Unfortunately, such a detailed presentation is outside our time limits here. Most of the methods discussed for diffusion and advection, however, are directly generalizable to Navier-Stokes equations.

The main goal of this section is to give a student enough understanding of the solution methods to read the manuals of software packages.

This section considers single phase, non-reacting incompressible and compressible flows without turbulence models.

4.1 Notations

During this section we move from the one dimensional problems to three dimensions. As the full sets of equations quickly become cumbersome, we will be using vector notations. Some of the equations are also collected to the Appendix A.2 using tensor notations. A quick introduction to the notations is given here.

Gradient, $\nabla \boldsymbol{u}$ Gradient tells the rate of change of the variable. Usually the change is relative to spacial coordinates. Gradient is often taken from both scalar and vector values. A familiar example is the heat flux from Eq. 2.1. Heat flux depends on the rate of change of temperature.

For a scalar ϕ

$$\mathrm{grad}(\phi) = \nabla\phi = \partial_{\mathrm{i}}\phi = \frac{\partial\phi}{\partial \mathrm{x_i}} = \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z}\right) \tag{4.1}$$

For a vector $\boldsymbol{u}$

$$\mathrm{grad}(\boldsymbol{u}) = \nabla\boldsymbol{u} = \partial_{\mathrm{i}}\mathrm{u_j} = \frac{\partial \mathrm{u_j}}{\partial \mathrm{x_i}} = \begin{pmatrix} \frac{\partial u_1}{\partial x} & \frac{\partial u_2}{\partial x} & \frac{\partial u_3}{\partial x} \\ \frac{\partial u_1}{\partial y} & \frac{\partial u_2}{\partial y} & \frac{\partial u_3}{\partial y} \\ \frac{\partial u_1}{\partial z} & \frac{\partial u_2}{\partial z} & \frac{\partial u_3}{\partial z} \end{pmatrix} \tag{4.2}$$

A gradient from a scalar produces a vector. A gradient of vector produces a tensor.

Divergence, $\nabla \cdot \boldsymbol{u}$ Divergence is one of the most common operators in fluid dynamics. In the context of finite volume method, it tells the difference of fluxes entering and leaving the control volume. For example, in the one dimensional heat conduction in Eq. 2.1, we take the divergence of heat fluxes.

Mathematically, divergence can be though as a dot product of gradient operator and the vector

$$\mathrm{div}\mathbf{u} = \frac{\partial u_i}{\partial x_i} = \nabla \cdot \mathbf{u} = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \cdot (u, v, w) = \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial x} + \frac{\partial u_3}{\partial x} \tag{4.3}$$

Divergence of a vector results in a scalar.

Laplacian Laplacian is a common operator in fluid dynamics because it closely relates to diffusion. Laplacian is a divergence of gradient

$$\text{laplacian}(\mathbf{u}) = \Delta\mathbf{u} = \nabla^2\mathbf{u} = \nabla \cdot \nabla\mathbf{u} \tag{4.4}$$

We have given this operator a name because it is so common. For example, in the one dimensional heat equation, Eq. 2.1, we have a Laplacian. Inside computer codes, Laplacian is usually also given a special treatment for speed and readability.

4.2 Finite volume method more mathematically

In the previous sections we avoided math when using finite volume method. We derived our discretized equations from balance equations for a control volume. We did this for simplicity, and because we will not have time to derive the disc

We will not derive discretized equations for the fluid flow because of time limits. We, however, need to be able to recognize the terms and have a rough understanding on how to discretize them. Here will will work through one dimensional examples.

Divergence, $\nabla \cdot \boldsymbol{u}$ The formal finite volume discretization start with taking a volume integral of the function. In one dimensions divergence is simply a gradient

$$\int_V \frac{\mathrm{d}u}{\mathrm{d}x}\mathrm{d}V \tag{4.5}$$

The volume integral is then changed to surface integral using the Gauss divergence theorem.

$$\int_V \frac{\mathrm{d}u}{\mathrm{d}x}\mathrm{d}V = \int_S \boldsymbol{n} \cdot \boldsymbol{u}\mathrm{d}S \tag{4.6}$$

where $\boldsymbol{n}$ is surface normal unit vector. The surface integral can further be divided as a sum operation for all the faces

$$\int_V \frac{\mathrm{d}u}{\mathrm{d}x}\mathrm{d}V = \int_S \boldsymbol{n} \cdot \frac{\mathrm{d}p}{\mathrm{d}x}\mathrm{d}S = \sum \boldsymbol{n} \cdot \boldsymbol{u}A \tag{4.7}$$

For one dimensional case this results in

$$\int_V \frac{\mathrm{d}u}{\mathrm{d}x}\mathrm{d}V = \int_S \boldsymbol{n} \cdot \frac{\mathrm{d}u}{\mathrm{d}x}\mathrm{d}S = \sum \boldsymbol{n} \cdot uA = u_e A_e - u_w A_w \tag{4.8}$$

which is similar to what we have been using this far. A three dimensional case would simply add more faces and terms.

Gradient, $\nabla \boldsymbol{u}$ In one dimensions, a gradient is the same thing as a divergence. The analysis applies.

Laplacian The Laplacian is a combination of divergence and gradient. A worked example in one dimension is already given in Section 2.1.1 for one dimensional heat conduction.

4.3 Governing equations

The governing equations are understood in a broad sense here to include the transport equations and fluid properties. We only discuss the equations here, the solution methods will be discussed later.

4.3.1 Continuity equation

Mass conservation is one of the key concepts in nature. For fluid flow, it translates into a continuity equation

$$\boxed{\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0} \tag{4.9}$$

For incompressible fluid density remains constant and Eq. 4.9 simplifies to

$$\boxed{\nabla \cdot \boldsymbol{u} = 0} \tag{4.10}$$

4.3.2 Momentum equations

Similarly to mass, momentum must also be conserved. Momentum equation for fluid flow reads

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = \nabla \cdot \boldsymbol{\tau} - \nabla p + \mathbf{f}_b \tag{4.11}$$

where for Newtonian fluids the shear rate tensor is

$$\begin{aligned} \boldsymbol{\tau} &= \mu \underbrace{\left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right)}_{2\mathbf{D}} + (-\frac{2}{3}\mu + \kappa) (\nabla \cdot \mathbf{u}) \mathbf{I} \\ &= 2\mu \mathbf{D} + (-\frac{2}{3}\mu + \kappa) (\nabla \cdot \mathbf{u}) \mathbf{I} \end{aligned} \tag{4.12}$$

where p, $\mathbf{f}_b$, μ, κ, and $\mathbf{D}$ are, pressure, body force, dynamic viscosity, bulk viscosity, and strain-rate, respectively.

We usually leave the terms $(-\frac{2}{3}\mu + \kappa) (\nabla \cdot \mathbf{u}) \mathbf{I}$ out as they are small for most subsonic flows. For incompressible flows $\nabla \cdot \mathbf{u} = 0$ and the terms a exactly zero. With these simplifications the momentum equation becomes

$$\boxed{\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = \nabla \cdot \left(\mu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right) \right) - \nabla p + \mathbf{f}_b} \tag{4.13}$$

For incompressible flow, this further simplifies to

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}\boldsymbol{u}) = \nabla \cdot (\nu (\nabla \mathbf{u})) - \frac{1}{\rho} \nabla p + \frac{\mathbf{f}_b}{\rho} \tag{4.14}$$

and with constant kinematic viscosity $\nu = \mu / \rho$ into

$$\boxed{\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}\boldsymbol{u}) = \nu \nabla^{\mathbf{2}} \boldsymbol{u} - \frac{1}{\rho} \nabla p + \frac{\mathbf{f}_b}{\rho}} \tag{4.15}$$

Note that there are great many slight variations of the momentum equations in the literature. Insignificant terms are often omitted without mention. Make certain you know witch version you need!

4.3.3 Energy equation

The energy equation states the energy conservation and reads

$$\frac{\partial \rho h}{\partial t}+\nabla\cdot(\rho \mathbf{u} h)+\frac{\partial \rho K}{\partial t}+\nabla\cdot(\rho \mathbf{u} K)-\frac{\partial p}{\partial t} = \nabla\cdot(\alpha\nabla h)+\nabla\cdot(\boldsymbol{\tau}\cdot\mathbf{u})+\rho r+\rho\mathbf{g}\cdot\mathbf{u} \quad (4.16)$$

where h, $K = \frac{1}{2} \mid \mathbf{u} \mid^2$, α, and r are, specific enthalpy, specific kinetic energy, thermal diffusivity, and specific heat source, respectively.

The energy equation is almost always simplified considerably. A common form reads

$$\boxed{\frac{\partial \rho h}{\partial t} + \nabla \cdot (\rho \mathbf{u} h) = \nabla \cdot (\alpha \nabla h) + S_h + \rho \mathbf{g} \cdot \mathbf{u}} \quad (4.17)$$

From Eq. 4.17 we can notice the similarity between the vector notation and the one dimensional problem we solved before. The first term $\frac{\partial \rho h}{\partial t}$ describe the changes in time. The second term $\nabla\cdot(\rho\mathbf{u}h)$ describes advection. The third term $\nabla\cdot(\alpha\nabla h)$ is conduction and S_h is heat source. The last term $\rho\mathbf{g}\cdot\mathbf{u}$ relates to work done against gravitation and is significant, for example, in natural convection.

4.3.4 General transport equation

As can be seen from the previous equations, most of the transport equations have a very similar form. They have the time derivative term, advection term, diffusion term, and source term. It is often convenient to use a general transport equation

$$\boxed{\frac{\partial \rho \phi}{\partial t} + \nabla \cdot (\rho \mathbf{u} \phi) = \nabla \cdot (\Gamma \nabla \phi) + S_\phi} \quad (4.18)$$

where ϕ is the transported variable and Γ is it's diffusion coefficient. General transport equation is most often used for passive scalars. A passive scalar is transported by the flow but does not affect the flow in any way. Examples of passive scalar use cases are small amounts of small particles or temperature.

4.3.5 Equation of state

Equation of state is needed to link density, pressure, and temperature together. The most common equation of state is the ideal gas equation

$$\begin{aligned} pV &= NR_uT \\ \Leftrightarrow \rho &= \frac{p}{RT} \end{aligned} \quad (4.19)$$

but the equation of state naturally depends on the fluid.

4.3.6 Fluid properties

If there are considerable temperature of pressure changes in the system, it is reasonable to expect significant changes in fluid properties. The most common fluid properties are the fluid viscosity ν or μ and heat diffusivity α. If we are running a incompressible fluid simulation we also may need the density.

Even though fluid properties are readily available for common fluids and can be changed inside the program with minimal user interaction, it is usually good to avoid varying fluid properties if possible. If the changes are small and do not affect the results, they may still make the solution much slower.

4.3.7 Additional physics

Great many addition can be made to the basic fluid flow problem considered here. Chemical reactions, multi component or phase flows, phase changes etc. are all readily added to the system.

Each layer of complication, however, adds to the computational cost. With complex systems it is also often hard to measure the accuracy of the model as there are so many parameters. It would be ideal to some how isolate or the parts of the simulation from each other for validation. In practice, this may be very difficult. It still a good practice to start as simple as possible.

4.4 Equation summary

Typical versions of the compressible and incompressible flow equations are summarized here. Make sure you understand the simplifications!

4.4.1 Compressible flow

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = \nabla \cdot \left(\mu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right) \right) - \nabla p + \mathbf{f}_b \tag{4.20}$$

$$\frac{\partial \rho h}{\partial t} + \nabla \cdot (\rho \mathbf{u} h) = \nabla \cdot (\alpha \nabla h) + S_h + \rho \mathbf{g} \cdot \mathbf{u} \tag{4.21}$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{4.22}$$

$$\rho = \frac{p}{RT} \tag{4.23}$$

4.4.2 Incompressible flow

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}\boldsymbol{u}) = \nu \nabla^{\mathbf{2}} \boldsymbol{u} - \frac{1}{\rho} \nabla p + \frac{\mathbf{f}_b}{\rho} \tag{4.24}$$

$$\nabla \cdot \boldsymbol{u} = 0 \tag{4.25}$$

4.5 Non-linear advection term

When comparing Eqs. 4.20-4.25 and the advection-diffusion equations considered before, we notice a lot of familiar terms.

The energy equation, Eq. 4.21, is the same equation as the one we have been dealing with in advection diffusion problems. Now it has just been written in vector form to allow three dimensions. We also have added the gravity related source term $\rho\mathbf{g}\cdot\mathbf{u}$.

The continuity equation, Eq. 4.22, consists of a time derivative terms and a divergence term. Both are familiar from heat examples before. The equation of the state, Eq. 4.23, is an algebraic equation and requires no special attention.

The differences arise from the non-linear advection term in momentum equations, Eq. 4.20. Expect for the momentum advection term $\nabla\cdot(\rho\mathbf{uu})$, all the terms in Eqs. 4.20-4.25 are linear for constant fluid properties.

The way to tackle the non-linear advection terms is straight forward, we linearize it. The details vary, but the basic idea is always to replace one of the velocities $\boldsymbol{u}$ with a known velocity $\hat{\boldsymbol{u}}$ from previous time step or iteration. The advection term becomes

$$\nabla\cdot(\rho\mathbf{uu}) \approx \nabla\cdot\left(\underbrace{\rho\hat{\mathbf{u}}}_{\text{advecting}}\ \overbrace{\boldsymbol{u}}^{\text{advected}}\right) \tag{4.26}$$

and after successful iteration process the two velocities are close enough. It is conventional to call the velocity $\hat{\boldsymbol{u}}$ advecting velocity and $\boldsymbol{u}$advected velocity. This convention arises from the analogy to general advection problem.

One important detail is that the advecting and advected velocities often need different discretization to produce a converging system. The advecting term is usually discretized with central differencing and the advected term with upwind or similar.

In most software packages there is only one discretization option for the advection term and this option means the discretization of the advected velocity. The advected term is often interpolated linearly and there might be a option called interpolation in the software package. Note that unless you are able to see the source code, details like these are often left untold.

Especially for the incompressible case, the advection term is often given in a non-conservative form $\boldsymbol{u}\cdot\nabla\boldsymbol{u}$. This form can be derived by using the product rule and continuity. Using the product rule, the time derivative and advection terms in momentum equation Eq. 4.20 becomes

$$\begin{aligned}
\frac{\partial\rho\mathbf{u}}{\partial t} + \nabla\cdot(\rho\mathbf{uu}) &= \underbrace{\rho\frac{\partial}{\partial t}\boldsymbol{u} + \boldsymbol{u}\frac{\partial}{\partial t}\rho}_{\text{time derivative}} + \underbrace{\rho\boldsymbol{u}\cdot\nabla\boldsymbol{u} + \boldsymbol{u}\nabla\cdot(\rho\boldsymbol{u})}_{\text{advection}} \\
&= \rho\left(\frac{\partial}{\partial t}\boldsymbol{u} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) + \boldsymbol{u}\underbrace{\left(\frac{\partial}{\partial t}\rho + \nabla\cdot(\rho\boldsymbol{u})\right)}_{\text{continuity=0}} \\
&= \rho\left(\frac{\partial}{\partial t}\boldsymbol{u} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right)
\end{aligned} \tag{4.27}$$

In this form, the linearization is usually done be taking the gradient from the known values as

$$\boldsymbol{u} \cdot \nabla \boldsymbol{u} \approx \boldsymbol{u} \cdot \nabla \hat{\boldsymbol{u}} \tag{4.28}$$

The practice of using interpolation for one of the terms $\nabla \hat{\boldsymbol{u}}$ and upwind like scheme for the other ($\boldsymbol{u}$) might be easier to see from this form.

4.6 Solution methods for compressible flow

Solving for compressible flow equations is more straight forward, even if not necessarily easier, that incompressible flow equations. We will first briefly discuss the compressible flow solution.

When solving for compressible flow with constant fluid properties like viscosity, we usually have six equations to solve and six variables. Therefore the system is closed and can be solved.

In practice, however, the solution easily becomes unstable or does not converge. There is no single best way for solving the equations. Instead, there are many known ways that often work in certain kinds of problems. Consult manuals and scientific papers on how to choose the correct method.

Usually, the momentum equations, Eqs. 4.20, are solved for for velocities and the energy equation, Eq. 4.21, for temperature.

The continuity equation can be solved directly for density. Then the solved temperature and density are used to solve for pressure using the equation of state.

It is, however, more common to solve the continuity equation for pressure in subsonic flows. This is often done using techniques similar to those used for incompressible flows. see Sec. 4.7.

Supersonic or transient flows require special methods to remain stable. The pressure shocks form discontinuous jumps into the solution. The governing equations 4.9 - 4.19 are perfectly capable to describe compressible flow, but the solution is more difficult than for subsonic flows.

4.7 Solution methods for incompressible flow

It is often beneficial to solve the incompressible versions of flow equations. For low Mach number flows, the density changes are often small, and will not affect the flow much.

As the changes in density are small, the equations for density become stiff and hard to solve numerically. As a rule of the thump, the compressibility effect can be ignored if Mach number $\mathrm{Ma} < 0.3$. This, of course, depends on the case. For example large temperature gradients may have significant effects on density.

Ignoring the fluid properties, we now have four equations and four unknown variables. The system is closed. The pressure, however, presents a problem.

Similarly to the compressible flow case, the momentum equations are used to solve for the velocities. After this we only have the continuity equation to use for pressure. With incompressible flow equation, however, the pressure is not present in the continuity equation.

The problem is usually solved by first solving the velocities from momentum equations with an old pressure field. This solution will not satisfy continuity.

The momentum equations are then substituted into the continuity equation. With suitable simplifications this yields a correction to pressure field and velocity field. The corrected fields will satisfy continuity.

The intuitive translation of the math is that, if too much material is flowing into a cell, increase the pressure. If too little material is flowing into a cell, decrease the pressure. And then fix the velocities accordingly.

The math operates on discretized equations and would take a long while to work through here. An explanation can be found, for example, from *Versteeg and Malalasekara 2007, Chapter 6.4 [6].*

The most common algorithm that operates in the explained manner is called SIMPLE. During the derivation of SIMPLE, some rather crude simplifications are made.

SIMPLER and SIMPLEC are very similar to SIMPLE but less simplification are made. This makes the SIMPLER and SIMPLEC more complicated and computationally expensive than SIMPLE, but also makes to usually converge in less steps.

PISO is another very similar algorithm to SIMPLE. PISO was originally developed for transient flows, but is often used for steady state solutions also.

All the methods should converge to the same values. The only differences are in derivation and convergence rates.

SIMPLE scheme and its relatives SIMPLEC and SIMPLER all require under relaxation to remain stable. Under relaxation is yet another word for linear interpolation. We interpolate the next solution from the newly calculated one, and the old solutions as

$$u = \alpha u_{new} + (1 - \alpha)\, u_{old} \tag{4.29}$$

where α is the under relaxation and should be chosen between zero and unity.

4.8 Pressure interpolation

In the momentum equations, we now have a new source terms, the pressure gradient, $-\nabla p$. This terms requires some special attention.

Historically, pressure decoupling was a major problem with fluid flow solutions. Pressure decoupling occurs when linear interpolation is used for pressure without any corrections. The decoupling is easily demonstrated with a simple example as shown in Fig. 4.1.

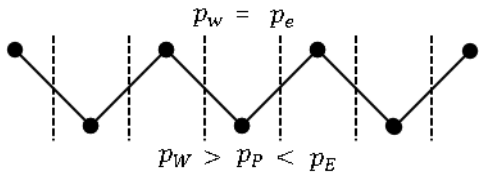


Figure 4.1: Pressure decoupling

With an oscillating pressure field with uniform mesh and $p_W = p_E$ the face pressures are interpolated equal and there will be no pressure source term $-\nabla p$.

There are two main strategies for tackling this problem. One is to use staggered grids, in which pressure and velocity is solved using different meshes. See Fig. 4.2 for an example.

This method is accurate as it requires no interpolation of fields. On the other hand it is more complex and requires more memory for the two meshes. As far as the author aware, the PRESTO! scheme in FLUENT uses a staggered grid approach. Historically, staggered grid was very important.

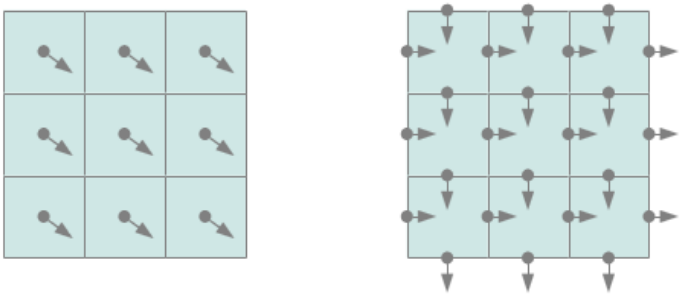


Figure 4.2: Staggered grid. Scalar variables on the left and vector variables on the right. [1]

The second, and more common, method is to apply a correction to the linearly interpolated pressure. The most famous correction is the Rhie and Chow interpolation method. The correction remove the decoupling and smooth oscillating pressure fields. For details, see for example [3].

The pressure velocity coupling and associated problems are the main reason we do not program our own fluid flow solver on this course. It is, however, possible to choose a practice case in which the pressure gradient is known or uniform. In this kind of problems the pressure-velocity coupling can be ignored.

4.9 Boundary conditions

All the fields in the governing equations require boundary conditions. We will limit ourself to the incompressible case here and only talk about pressure and velocity boundary conditions.

There are great many different boundary conditions for different cases. We will only consider two primitive boundary condition, the fixed value and zero gradient, here.

Many more complicated boundary conditions are similar to at least one of these. For example, a constant mass flow boundary condition is very similar to fixed velocity boundary condition, but allows the velocity profile to develop into a more physically reasonable form. Unless you actually know the velocity profile, it is usually better to use a constant mass flow boundary condition to fixed velocity to avoid non-physical solution near the boundary.

One should never define a fixed value boundary condition for both the velocity and pressure at the same boundary. This combination is very unstable.

It is usually preferable to define fixed velocity, or similar, boundary condition to the upstream boundary. Information moves much faster downs stream in the

flow and exploiting this results in faster convergence. Pressure information travels instantly in incompressible flow and very fast in compressible flow.

At least for the author, non-sense boundary conditions are the most common reason for crashing simulations.

4.10 Coupled and segregated solvers

Most often, the governing equations are solved individually, in series. This is called a segregated solution. A segregated solution for a compressible flow problem would be something like the following

1. Solve linearized x-momentum equation for u. $A_u u = b_u$
2. Solve linearized y-momentum equation for v. $A_v v = b_v$
3. Solve linearized z-momentum equation for w. $A_w u = b_w$
4. Solve energy equation for T. $A_T T = b_T$
5. Solve continuity equation for ρ. $A_\rho \rho = b_\rho$
6. Solve pressure from equation of state.
7. Repeat steps 1-6 until convergence is reached.

The iteration typically takes a lot of steps unless a very good initial guess is available. If we are solving a transient case, the previous time step serves as a good initial guess and the iteration is much faster.

It is also possible to combine all or some of the solution steps into a single step. If we were, for example, to solve all the velocity components, the temperature and the density in a single step the resulting linear system would be similar to (not complete)

$$\begin{bmatrix} [A_u] & & & & & \\ & [A_y] & & & & \\ & & [A_w] & & & \\ & & & [A_T] & & \\ & & & & [A_\rho] \end{bmatrix} \begin{bmatrix} [u] \\ [v] \\ [w] \\ [T] \\ [\rho] \end{bmatrix} = \begin{bmatrix} [b_u] \\ [b_v] \\ [b_w] \\ [b_T] \\ [b_\rho] \end{bmatrix} \tag{4.30}$$

where each term inside [] is a matrix or vector similar to that in the segregated solution. Note that now there would be also extra coefficients linking, for example velocity x-component to all the other velocities and pressure. **These extra terms are not shown in the matrix above.**

The resulting system is much larger than the ones used in segregated solution. It requires more memory and takes longer to solve. Depending on the problem, a coupled solution *may* be much faster than segregated one. Or not. Again, the correct choice depends on the case.

4.11 Parallel solvers

In modern computational fluid dynamics, we often solve very large systems. Some vary basic understanding on parallel computation is in order.

Different solvers operate differently, and the following mostly mostly considers OpenFOAM. Other finite volume method based solvers are probably similar.

Parallelization of the linear systems solvers is difficult and usually not done. Parallelization is done on the mesh level by dividing the mesh into parts. Each part is then solved separately, in separate matrices, on different cores. The mesh interfaces between mesh parts is handled with interfaces similar to boundary conditions.

The parallelization at mesh level provides a very simple and effective parallelization in large systems. If you use OpenFOAM to solve a large system with 100 cores, it will solve it near 100 faster than only using one core. The key word, however, is *large*.

In order for information to flow from one mesh part to another, there must be communication between the parts. If there are a lot of parts in a small system, there will be a lot of communication compared to the actual computation. This communication can choke the solver. Therefore, it is not reasonable to always use as many cores as available.

It is not possible to know how how a certain case parallelizes without testing. As a rule of the thump, each core should have at least 50000 cells. Note that this number may be orders of magnitudes off in some cases.

Other parallelization methods also exists.

The previous analysis only applies to parallelization in space. In time, parallelization is not of much use. The previous time step must be known before we can proceed to the next one.

Part I
TODO

Everything from here on is very much unfinished. If you wish to read it anyway, expect all kinds of errors and changes afterwards.

5 Turbulence

6 Mesh

Appendix

A Math

Useful math that did not fit in the main text is collected here.

A.1 Tensor notation

Also known as index notation, Einstein notation, or Einstein summation convention.

If an **index appears twice in a some term, it is summed**

$$\frac{\partial u_j}{\partial x_j} = \sum_{j=1}^{3} \frac{\partial u_j}{\partial x_j} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \tag{A.1}$$

the sum operator $\sum$ is usually omitted. Sometimes a more compact notation is used as

$$\partial_j u_j = \frac{\partial u_j}{\partial x_j} \tag{A.2}$$

As the equations become longer and more complex, the value of short notations increase.

If an **index does not appear twice in any of the terms, it form different equations**

$$\frac{\partial p}{\partial x_i} = \partial_i p = \begin{matrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \\ \frac{\partial p}{\partial z} \end{matrix} \tag{A.3}$$

Combining the two rules

$$u_j \frac{\partial u_i}{\partial x_j} = u_j \partial_j u_i = \begin{matrix} u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} \\ u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} \\ u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} \end{matrix} \tag{A.4}$$

Note that if an index appears twice in some term, it is summed in all terms.

Considering, for example, momentum equation from the incompressible NS

$$u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j^2} \tag{A.5}$$

The index j appears twice in the advection term $u_j \frac{\partial u_i}{\partial x_j}$ and all term where it appears are summed. Index i appears only once and we need three different equations. Expanding Eq. A.5 leads to

$$
\begin{aligned}
u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z} &= -\frac{1}{\rho}\frac{\partial p}{\partial x}+\nu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right)\\
u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z} &= -\frac{1}{\rho}\frac{\partial p}{\partial y}+\nu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2}\right)\\
u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z} &= -\frac{1}{\rho}\frac{\partial p}{\partial z}+\nu\left(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2}\right)
\end{aligned}
\tag{A.6}
$$

A.2 Navier-Stokes with tensor notations

$$
\frac{\partial \rho u_i}{\partial t}+\frac{\partial\left(\rho u_j u_i\right)}{\partial x_j}=\frac{\partial \tau_{ij}}{\partial x_j}-\frac{\partial p}{\partial x_i}+f_i \tag{A.7}
$$

$$
\tau_{ij}=\mu\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_j}\right)+\left(-\frac{2}{3}\mu+\kappa\right)\frac{\partial u_j}{\partial x_j}\delta_{ij} \tag{A.8}
$$

$$
=2D_{ij}\mu+\left(-\frac{2}{3}\mu+\kappa\right)\frac{\partial u_j}{\partial x_j}\delta_{ij} \tag{A.9}
$$

$$
\frac{\partial \rho u_i}{\partial t}+\frac{\partial\left(\rho u_j u_i\right)}{\partial x_j}=\frac{\partial}{\partial x_j}\left(\mu\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_j}\right)\right)-\frac{\partial p}{\partial x_i}+f_i \tag{A.10}
$$

$$
\frac{\partial u_i}{\partial t}+\frac{\partial\left(u_j u_i\right)}{\partial x_j}=\frac{\partial}{\partial x_j}\left(\nu\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_j}\right)\right)-\frac{1}{\rho}\frac{\partial p}{\partial x_i}+f_i \tag{A.11}
$$

A.3 Common operators

A.3.1 Gradient

For a scalar ϕ

$$
\mathrm{grad}(\phi)=\nabla\phi=\partial_{\mathrm{i}}\phi=\frac{\partial\phi}{\partial \mathrm{x_i}}=\left(\frac{\partial\phi}{\partial x},\frac{\partial\phi}{\partial y},\frac{\partial\phi}{\partial z}\right) \tag{A.12}
$$

For a vector $\boldsymbol{V}$

$$
\mathrm{grad}(\boldsymbol{V})=\nabla\mathrm{V}=\partial_{\mathrm{i}}\mathrm{V_j}=\frac{\partial \mathrm{V_j}}{\partial \mathrm{x_i}}=\begin{pmatrix}\frac{\partial V_1}{\partial x} & \frac{\partial V_2}{\partial x} & \frac{\partial V_3}{\partial x}\\ \frac{\partial V_1}{\partial y} & \frac{\partial V_2}{\partial y} & \frac{\partial V_3}{\partial y}\\ \frac{\partial V_1}{\partial z} & \frac{\partial V_2}{\partial z} & \frac{\partial V_3}{\partial z}\end{pmatrix} \tag{A.13}
$$

A.3.2 Divergence

For a vector $\boldsymbol{V}$

$$
\mathrm{div}\left(\boldsymbol{V}\right)=\nabla\cdot\boldsymbol{V}=\partial_j V_j=\frac{\partial V_j}{\partial x_j}=\frac{\partial V_1}{\partial x}+\frac{\partial V_2}{\partial y}+\frac{\partial V_3}{\partial z} \tag{A.14}
$$

A.3.3 Laplacian

For a scalar ϕ

$$\text{laplacian}\,(\phi) = \Delta\phi = \nabla^2\phi = \partial_{\text{j}}^2\phi = \frac{\partial^2\phi}{\partial x_j^2} = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} \qquad \text{(A.15)}$$

A.4

B External Resources

B.1 Books etc.

- **Recommonded for this course**
  - An Introduction to Computational Fluid Dynamics: The Finite Volume Method, Versteeg, H.K. and Malalasekera, W. 2007
- FVM diffusion in Wikipedia
  - https://en.wikipedia.org/wiki/Finite_volume_method_for_one-dimensional_steady_state_diffusion
  - https://en.wikipedia.org/wiki/Finite_volume_method_for_two_dimensional_diffusion_problem
  - https://en.wikipedia.org/wiki/Finite_volume_method_for_three-dimensional_diffusion_problem
- For advanced studies in OpenFOAM
  - https://www.researchgate.net/profile/Tobias_Holzmann/publication/307546712_Mathematics_Numerics_Derivations_and_OpenFOAMR/links/59c7ffde0f7e9bd2c014693c/Mathematics-Numerics-Derivations-and-OpenFOAMR.pdf

B.2 Programs, programming, libraries etc.

- https://anaconda.org/
- http://www.openfoam.org/
- http://www.openfoam.com/
- http://www.ansys.com/
- https://fenicsproject.org/
- https://www.ctcms.nist.gov/fipy/index.html

References

[1] grille normal et staggered-grid, 2012. [Online; accessed 20-Feb-2018] CC BY-SA 3.0.

[2] Dominique Aegerter. Aerodynamics of motorcycles using cfd simulations, 2016. [Online; accessed 12-Jan-2018] CC BY-SA 4.0.

[3] Fabian Peng Karrholm. Rhie-chow interpolation in openfoam, 2006. [Online; accessed 19-Feb-2018].

[4] R. Pitz and J. Daily. Experimental study of combustion in a turbulent free shear layer formed at a rearward facing step. 1981.

[5] Turo Valikangas, 2018. Personal commmunication.

[6] H.K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education Limited, 2007.

[7] WinnerMotors. [Online; accessed 23-Jan-2018] CCO.