# Numerical Techniques for Process Modelling
# Part I:
# Finite Volume Method
# and
# Computational Fluid Dynamics

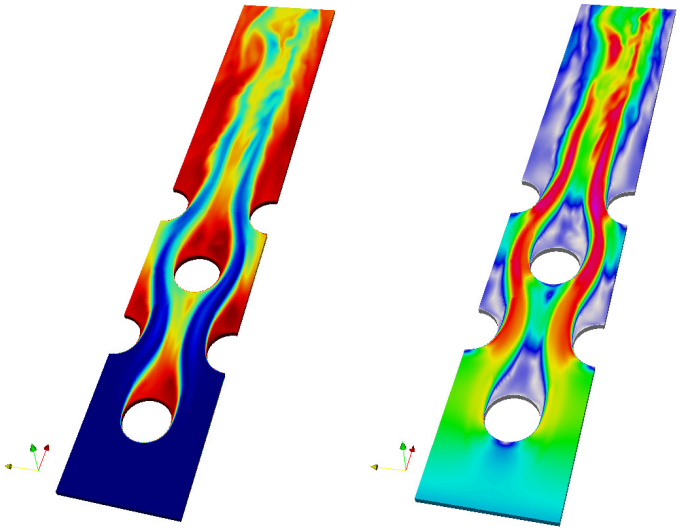
KEB-45250
Numerical Techniques for Process Modelling,
Prosessien numeerinen mallinnus
5 credits

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

Version 0.1

Updated on:

February 6, 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 | Chapters 1-2 | | |

# 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 [4]. 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. A typical mesh contains hundreds of thousands or millions of cells.

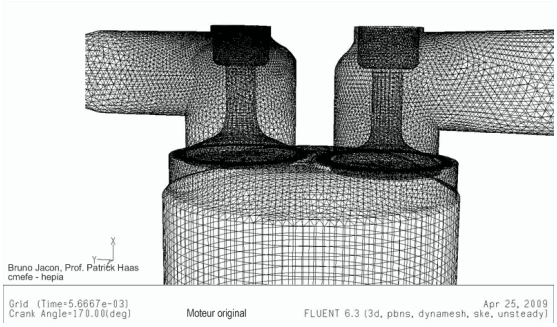



Figure 1: Engine mesh[1]

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. 2. The velocity distribution is, of course, continuous. Discretizing the profile with ten cells gives the discretized profile in Fig. 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. 3.

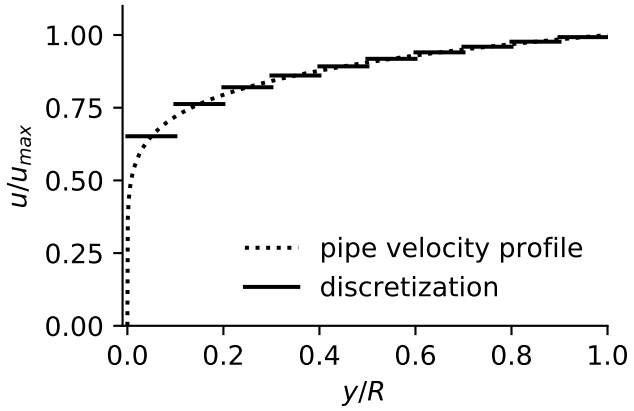



Figure 2: Fully developed turbulent mean velocity profile in a pipe

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. 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. 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. 3 and unstructured one elsewhere.

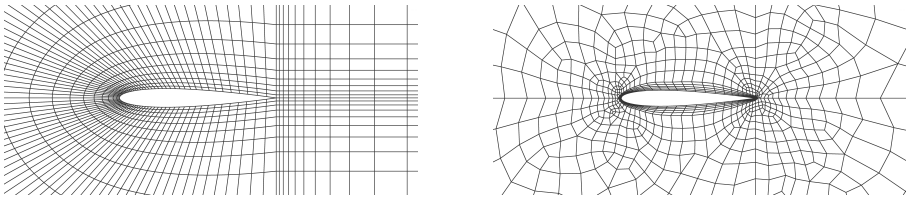


Figure 3: Structured and unstructured (low quality) meshes of a 2D wing

The cells in meshes may have an arbitrary shape. Simple, regular shapes like the ones in Fig. 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.

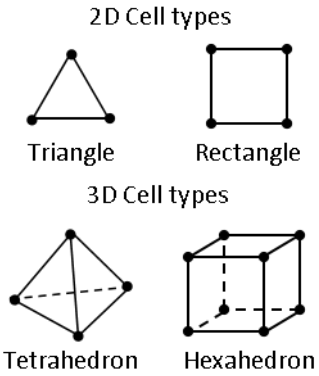



Figure 4: Cell types

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.

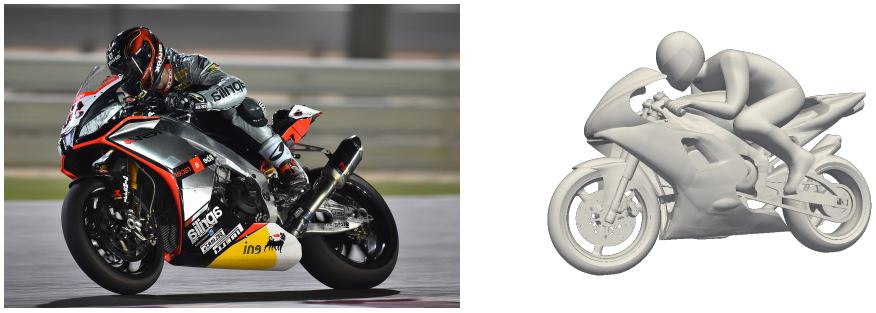


Figure 5: Geometry simplification[5]

In Fig. 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.

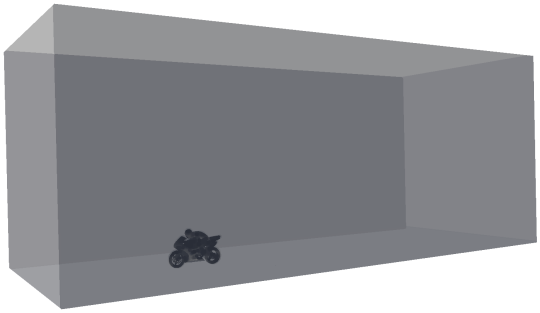


Figure 6: Calculation domain

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. 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. 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.

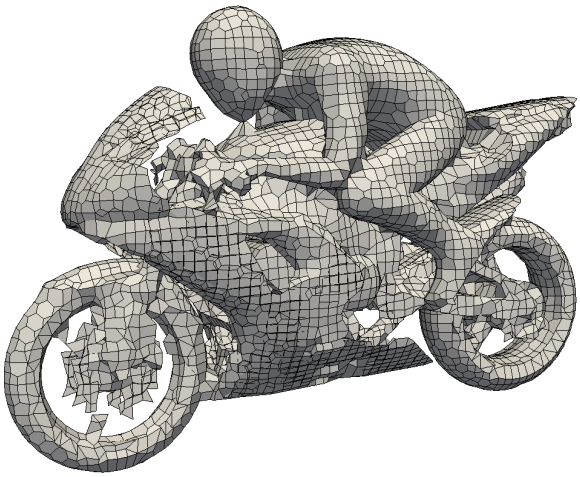


Figure 7: Surface mesh (low quality)

The mesh must have sufficiently small cells to accurately represent the modeled phenomena. In Fig. 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. 8.

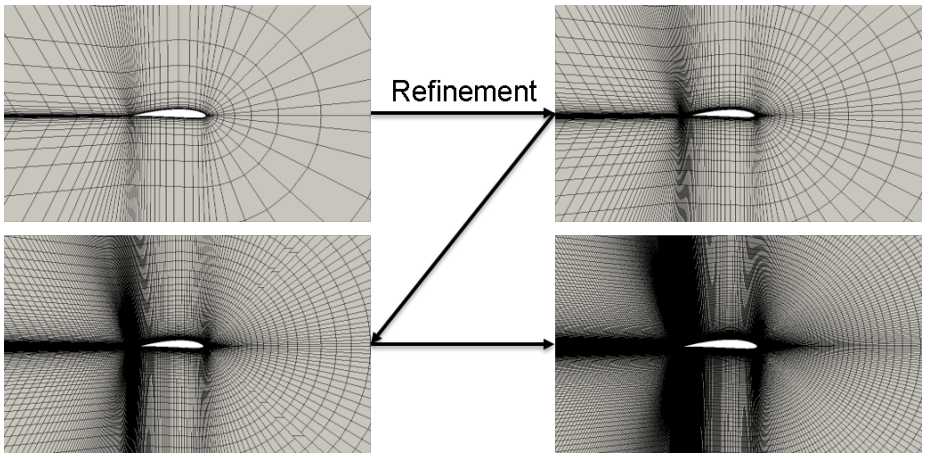



Figure 8: Mesh refinement

Refining the mesh with the same meshing strategy, as is done in Fig. 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. 9 for examples of boundary conditions. The mathematical definitions will be given later.

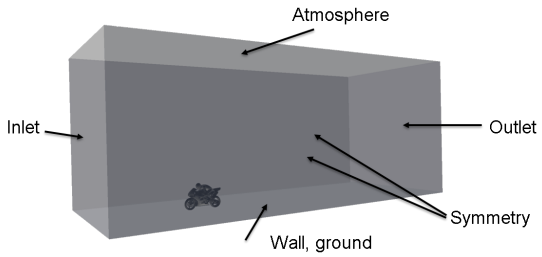



Figure 9: Boundary conditions

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 [2], see Figs. 10 and 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. 10 and 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.

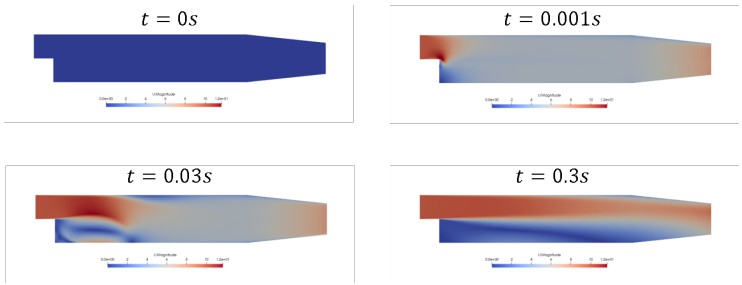



Figure 10: Transient solution

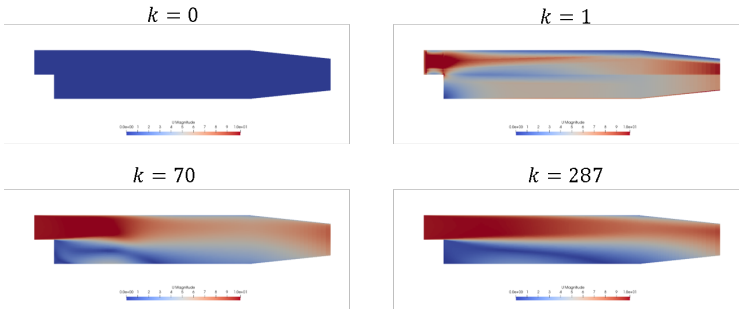



Figure 11: Steady solution

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. 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.

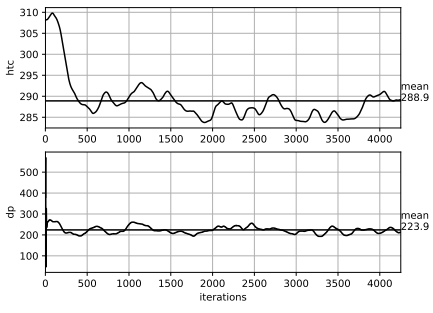



Figure 12: Pseudo convergence of steady state solution

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. 13. It would be next to impossible the measure the velocity and temperature profiles inside the device and visualize them.

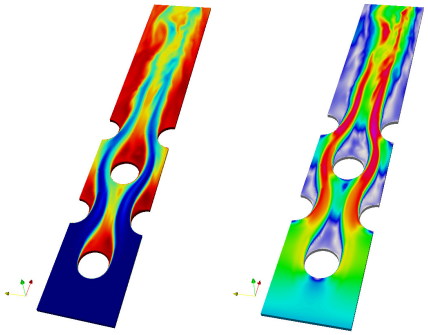


Figure 13: Temperature and velocity fields at one time step[3]

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. [4]*

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 [4] 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. 14. 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. 14.

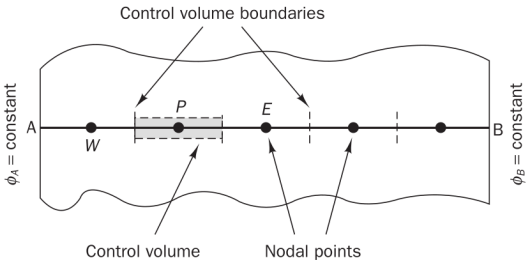



Figure 14: Control volumes and nodes [4]

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. 16. The cell size, or distance from one face to another is called $\Delta x$ as shown in Fig. 16.

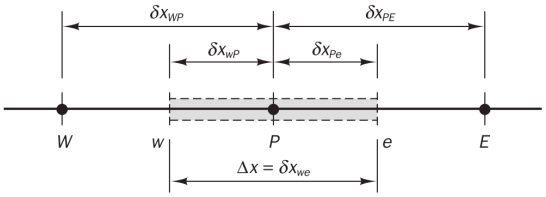



Figure 15: Faces and distances [4]

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.

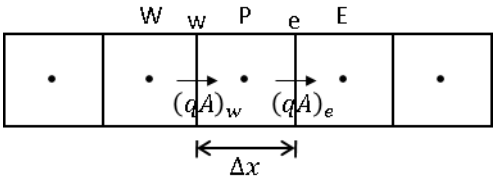



Figure 16: Heat conduction

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. 17 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. 17.

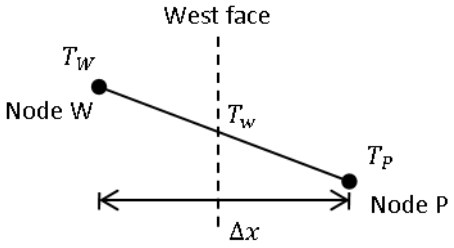



Figure 17: Central differencing

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 $\Delta x$ 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

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

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

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 $\Delta x$ 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.

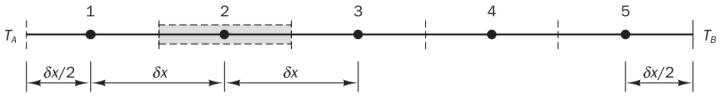



Figure 18: Boundary conditions 1D

The cells that touch boundaries are called boundary cells and require special attention. Let us consider cell 1 in Fig. 18.

### 2.2.1 Constant temperature boundary

*See Versteeg and Malalasekare [4] 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. 18 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 [4] 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 19: 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 $T_B$ 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. 19. 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()
```

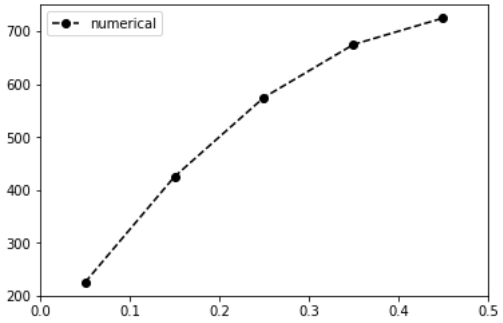



Figure 20: Plotting

## 2.4 Unsteady

*See Versteeg and Malalasekare [4] 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 $\rho$ 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 [4].*

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 $\Delta t$ 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. 21. This often results in simpler and faster code.

To progress the solution repeat the process in a loop. An example is given in Fig. 21.

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

Figure 21: Time stepping

## 2.5 2D and 3D cases

*See Versteeg and Malalasekare [4] 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. 22.

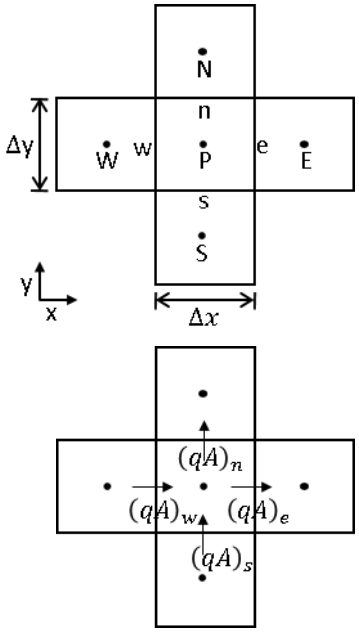



Figure 22: 2D notations and control volume

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. 23. 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 23: 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 $\phi$ 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 $\rho$ 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.

# 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.

## 3 Advection-diffusion

## 4 Navier-Stokes

### 4.1 The NS equation

### 4.2 Compressible vs. incompressible

### 4.3 Pressure coupling

### 4.4 Boundary conditions

## 5 Mesh

## 6 Turbulence

# 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 Common operators

### A.2.1 Gradient

For a scalar $\phi$

$$
\mathrm{grad}(\phi) = \nabla\phi = \partial_{\mathrm{i}}\phi = \frac{\partial\phi}{\partial \mathrm{x_i}} = \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z}\right) \tag{A.7}
$$

For a vector $\boldsymbol{V}$

$$
\mathrm{grad}(\boldsymbol{V}) = \nabla \mathrm{V} = \partial_{\mathrm{i}}\mathrm{V_j} = \frac{\partial \mathrm{V_j}}{\partial \mathrm{x_i}} = \begin{pmatrix} \frac{\partial V_1}{\partial x} & \frac{\partial V_2}{\partial x} & \frac{\partial V_3}{\partial x} \\ \frac{\partial V_1}{\partial y} & \frac{\partial V_2}{\partial y} & \frac{\partial V_3}{\partial y} \\ \frac{\partial V_1}{\partial z} & \frac{\partial V_2}{\partial z} & \frac{\partial V_3}{\partial z} \end{pmatrix} \tag{A.8}
$$

### A.2.2 Divergence

For a vector $\boldsymbol{V}$

$$
\mathrm{div}\,(\boldsymbol{V}) = \nabla \cdot \boldsymbol{V} = \partial_j V_j = \frac{\partial V_j}{\partial x_j} = \frac{\partial V_1}{\partial x} + \frac{\partial V_2}{\partial y} + \frac{\partial V_3}{\partial z} \tag{A.9}
$$

### A.2.3 Laplacian

For a scalar $\phi$

$$
\mathrm{laplacian}\,(\phi) = \Delta\phi = \nabla^2\phi = \partial_{\mathrm{j}}^2\phi = \frac{\partial^2\phi}{\partial x_j^2} = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} \tag{A.10}
$$

## A.3

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

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

[3] Turo Valikangas, 2018. Personal commmunication.

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

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