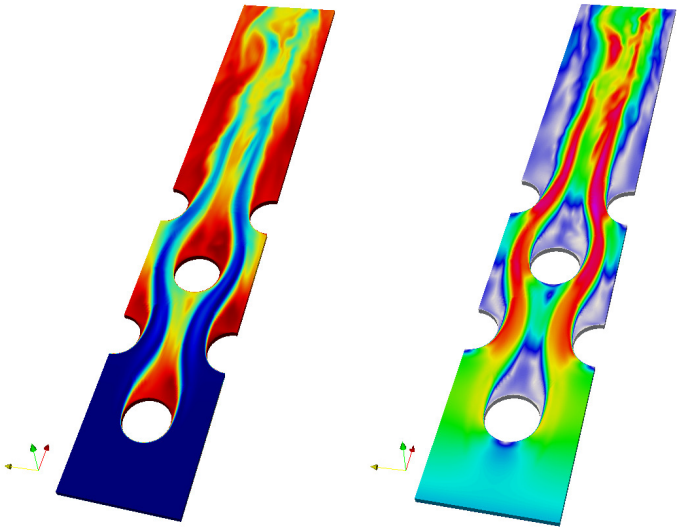
Numerical Techniques for Process Modelling
Part I:
Finite Volume Method
and
Computational Fluid Dynamics

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

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

Updated on:

January 26, 2018



License

Publication version history

These notes are written during the first implementation of this course. 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	?	Chapters 1-?		

Contents

1 Introduction **2**
- 1.1 What is CFD? . . . 2

2 OLDIntroduction **8**
- 2.1 What is CFD? . . . 8
- 2.2 Work flow . . . 9
  - 2.2.1 Pre-processing . . . 10
  - 2.2.2 Solver . . . 13
  - 2.2.3 Post-processing . . . 14
- 2.3 Convergence . . . 15
- 2.4 Mesh independency . . . 16
- 2.5 Turbulence . . . 16
- 2.6 Navier-Stokes equation . . . 16
- 2.7 Other equations . . . 16

3 Heat Conduction (diffusion) **17**
- 3.1 Developing the equations for 1D problem . . . 17
  - 3.1.1 Same thing in math . . . 20
- 3.2 Boundary conditions . . . 21
  - 3.2.1 Constant temperature boundary . . . 21
  - 3.2.2 Zero gradient (insulated) boundary . . . 22
- 3.3 Matrix assembly . . . 22
- 3.4 Unsteady . . . 26
- 3.5 2D and 3D cases . . . 27
- 3.6 Notations . . . 29
- 3.7 Extra material . . . 29
  - 3.7.1 Performance issues with example codes . . . 29
  - 3.7.2 Application to other physics . . . 30

I TODO **32**

4 Advection-diffusion **32**

5 Navier-Stokes **32**
- 5.1 The NS equation . . . 32
- 5.2 Compressible vs. incompressible . . . 32
- 5.3 Pressure coupling . . . 32
- 5.4 Boundary conditions . . . 32

6 Mesh **32**

7 Turbulence **32**

A Math **33**
- A.1 Tensor notation . . . 33
- A.2 Common operators . . . 34
  - A.2.1 Gradient . . . 34
  - A.2.2 Divergence . . . 34

A.2.3 Laplacian 34
A.3 34

B External Resources **35**
B.1 Books etc. 35
B.2 Programs, programming, libraries etc. 35

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. If you prefer a full text book, see the introductory level text book by Versteeg and Malalasekera, An introduction to Computational Fluid Dynamics [5]. More detailed pointers to relevant section will be given in the text as well as other outside sources.

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. 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 note 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	

Structure

In Introduction 1 I attempt to give a overview of all the topics covered on this course. I will use minimal math. If you prefer equations, this section will be of little value to you.

1 Introduction

Computational fluid dynamics (CFD) offers great flexibility for an engineer. It can can give valuable insights into many problems outside the scope of analytical methods. Unlike analytical methods, CFD is not limited to simple geometries and can model most geometries encountered by an engineer. Varying material properties are also straight forward to model. 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 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 What is CFD?

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 pieces called cells. The cells form a mesh that fill the computational domain, see Fig. 1.

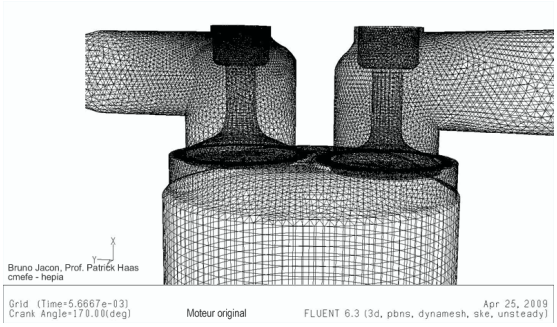


Figure 1: Engine mesh[1]

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. In analytical methods 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 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.

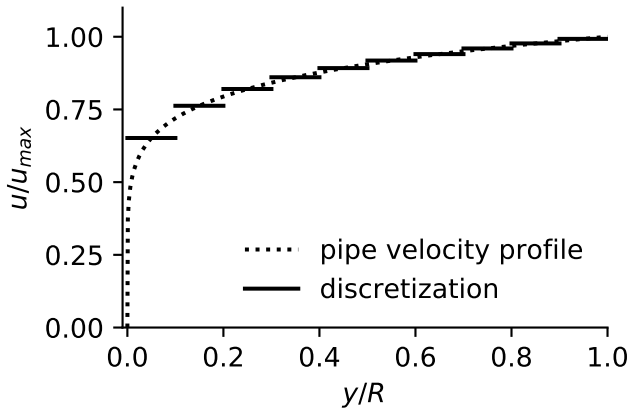


Figure 2: Fully developed turbulent velocity profile in a pipe

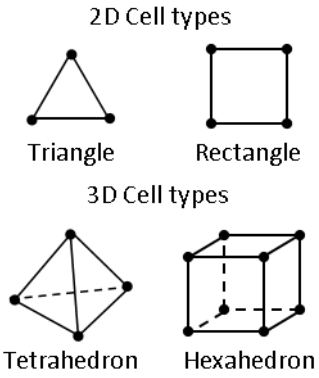


Figure 3: Cell types

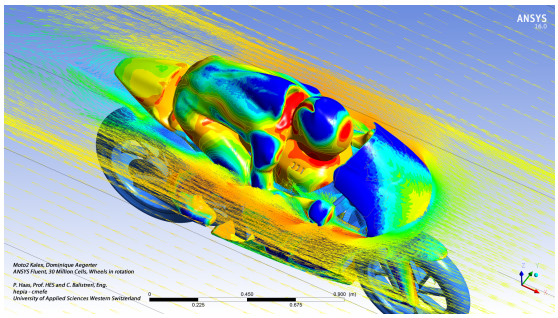


Figure 4: Aerodynamics of a motorcycle [1]

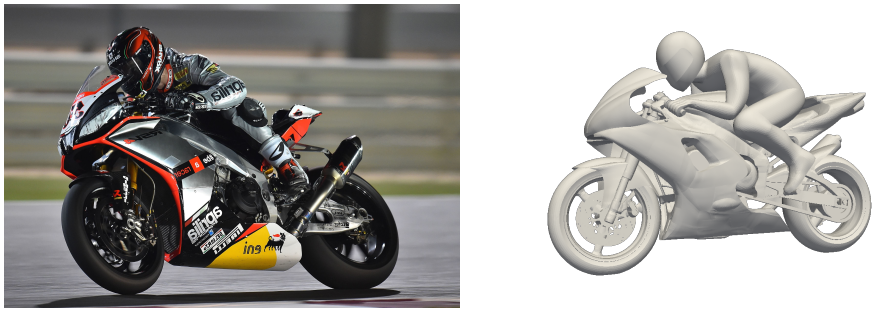


Figure 5: Geometry simplification[6]

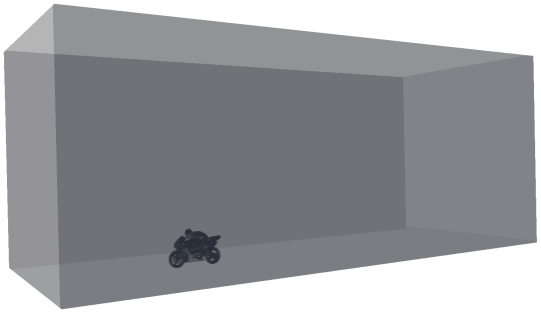


Figure 6: Calculation domain

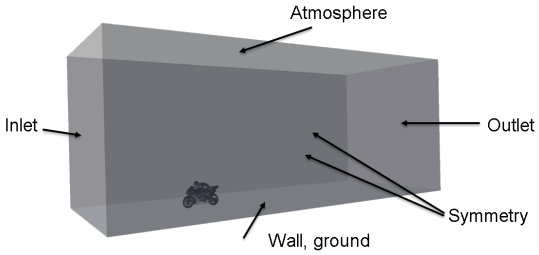


Figure 7: Boundary contions

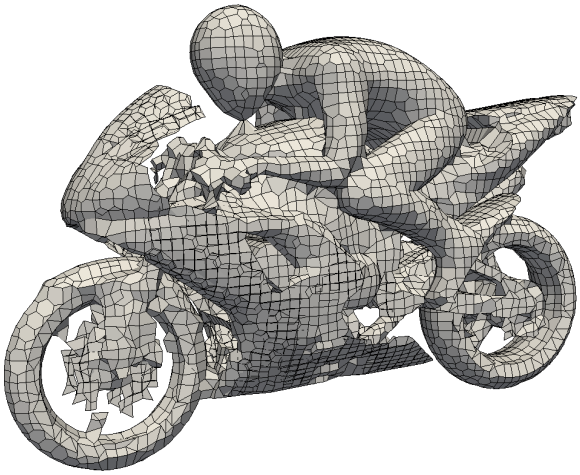


Figure 8: Surface mesh (low quality)

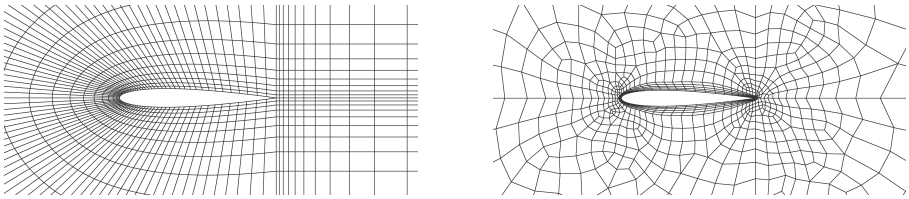


Figure 9: Structured and unstructured (low quality) meshes of a 2D wing

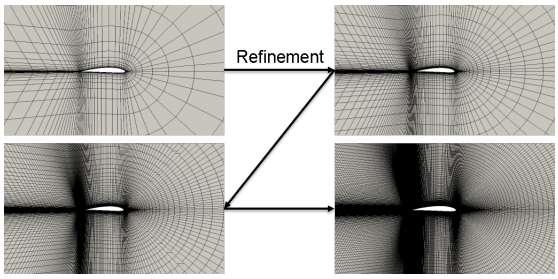


Figure 10: Mesh refinement

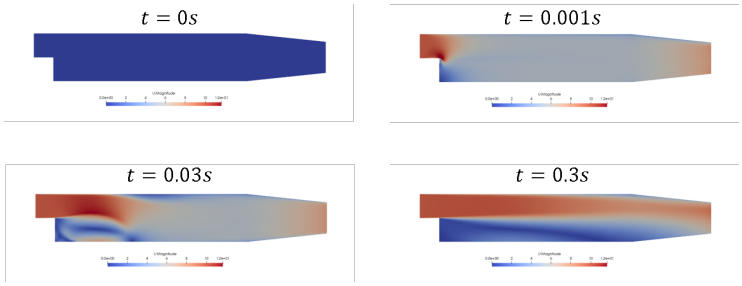


Figure 11: Unsteady solution

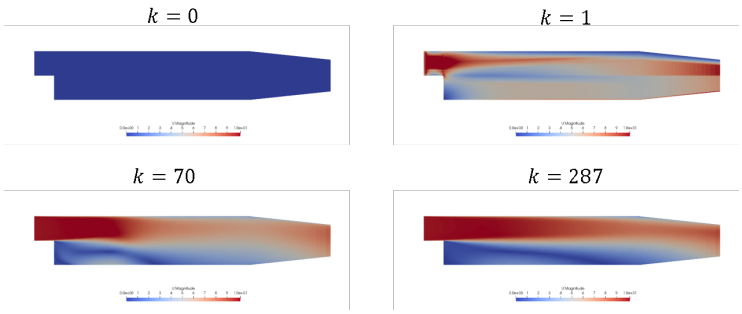


Figure 12: Steady solution

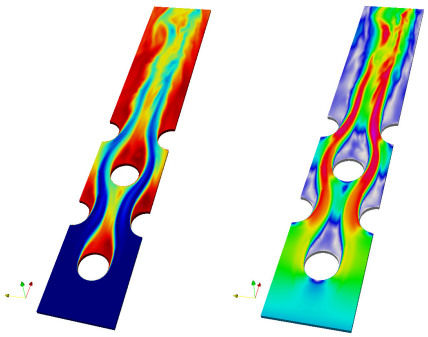


Figure 13: Temperature and velocity fields at one time step[4]

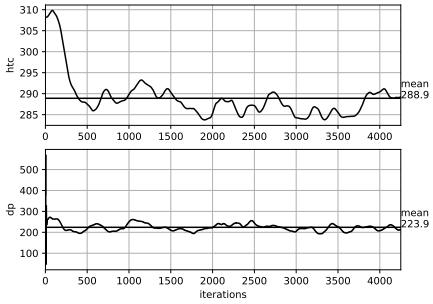


Figure 14: Pseudo convergence of steady state solution

2 OLDIntroduction

This section attempts to give on overview of the Computational Fluid Dynamics material discussed on this course. Math is avoided and all points will be returned to in greater detail.

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 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. Don't blame the program if you don't know what it's doing.

Computational Fluid Dynamics, CFD, has few fundamental limitations for an engineer. The underlying physics is usually well enough understood to give a near perfect representation of reality. The computational power to directly model the accurate physics are, however, not available to industry. At the moment the direct method, called Direct Numerical Simulation (DNS), is purely a research tool available only for a few turbulence researchers. In practical problems, we usually have to rely on computer resources saving models. Understanding these models is what makes a CFD expert.

This course will provide you with the basics of CFD. You will be familiar with the main ideas, able to program your own solvers for heat conduction cases, use existing CFD programs in simple cases, and learn more on your own.

2.1 What is CFD?

Probably the best place to start studying Computational Fluid Dynamics (CFD) is to go to Youtube and search with: CFD, Fluent, or OpenFOAM. There should be plenty of instructional videos.

In industry, the problem geometry is often complicated, see Fig. Fig. 4. , fluid properties chance in the domain or chemical reactions take place. Any of these is usually a big problem using analytical methods. With CFD, however, all of these problems are relatively straight forward to deal with. Fluid properties and chemical reactions can be calculated locally and complicated geometries are a matter of meshing.

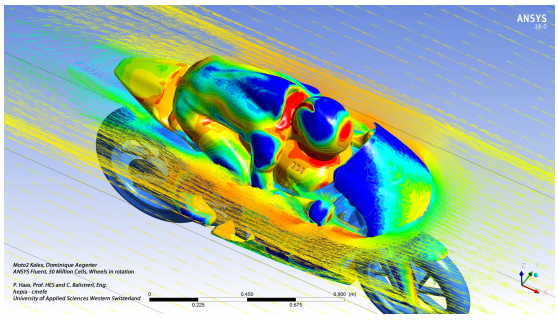


Figure 15: Aerodynamics of a motorcycle [1]

Meshing is a key concept in most CFD methods. We will only consider finite

volume method here and use it near synonymous to CFD. With meshing, we divide the calculation domain in a large number of small control volumes, see Fig. 1.

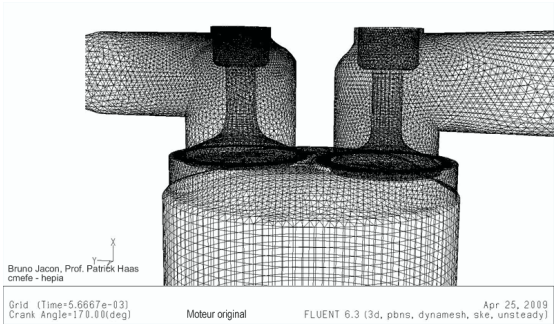


Figure 16: Engine mesh[1]

Meshing is useful because it allows us to solve many simple problems instead of one complex problem. Each cell and it's immediate neighbors form a local system for which balance equations ably, see Fig. 17. If the net mass flow $\sum \dot{m}_i$ from neighboring cells is positive, the mass inside the cell in the center must increase. We only need the information from the neighboring cells to do this. Similar balance equations are formed for momentum etc.

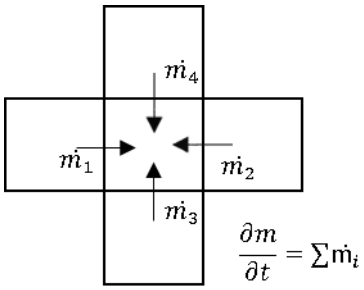


Figure 17: Cell

The ability to divide the large problem into small problems is the essence of numerical solution.

2.2 Work flow

The first thing to do with any problem is the understanding of relevant physical or chemical phenomena. It is usually beneficial to do some rough calculations using known correlations or simple energy balances to get a rough idea of what to expect. Every type of problem is different and experience plays a large part in how good the model will eventually be. If you have no previous experience it is best to try to find a published paper on the subject or ask somebody.

Once we are familiar with the problem the actual CFD work begins. The work flow is usually divided in three parts: pre-processing, solving, and post-processing.

2.2.1 Pre-processing

Pre-processing is everything that needs to be done before the actual numerical solver is used. The two main components are geometry processing (CAD) and meshing.

Geometry is the staring point of CFD simulation. There is basically two different options: you draw the geometry by yourself or you get the geometry from someone else. Either way, you must pay attention to the same things. What do you want to include to your model? See example in Fig. 18. On the left there is a photo of a real truck and on the right a simplified model for CFD.

Let us assume that the problem is to solve air resistance of the truck at a certain speed. For this calculation we need a simplified outer surface of the truck. Nothing in the inside is of interest. Small details like bolts are also meaningless for the air resistance and would make meshing much more complicated. One must usually make some educated guesses about what ti include in the model. If we are given a detailed CAD file of the truck we want to remove all the unimportant details .

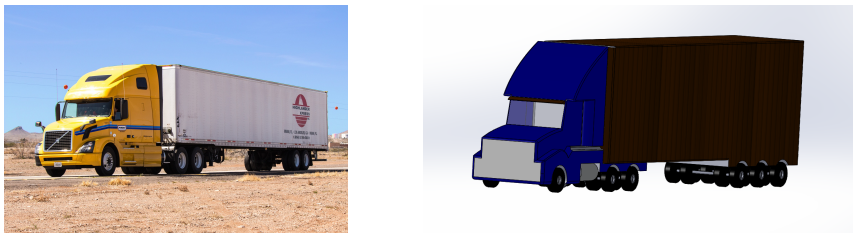


Figure 18: Geometry simplification[2, 3]

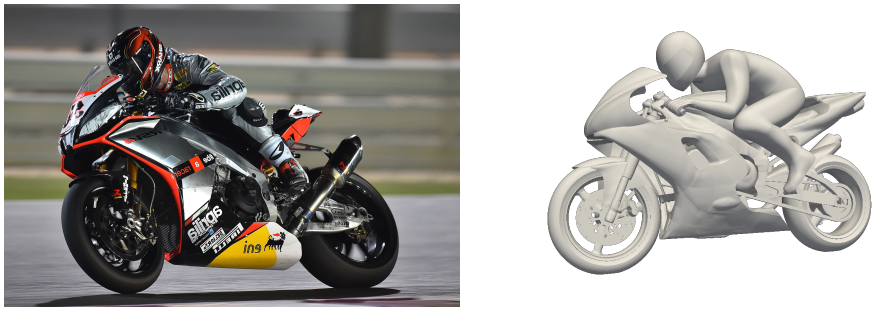


Figure 19: Geometry simplification[6]

Once the truck CAD model is ready, we need to model the actual calculation domain. For a air resistance calculation the domain would be something like shown in Fig. 6.

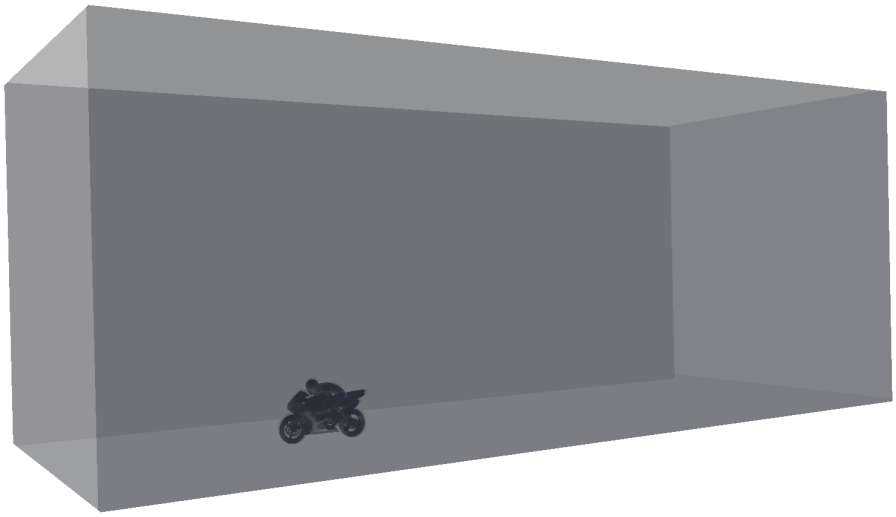


Figure 20: Calculation domain

We'll want to model the air around the truck. Not the truck itself. We need to model some air in front of the truck and to its sides. Most of the domain will, however, be behind the truck where the eddies form. The volume limited by the truck surface is cut away from the domain.

Meshing is the next step after calculation domain is created. Meshing is typically the most time consuming part of setting up a CFD simulation. The actual solution may run longer but requires no user input. The mesh must be fine enough to be apple to represent a meaningful solution of the problem. One of the main concepts behind Finite Volume Method, the numerical method used on this course, is that each cell only contain one value for a given quantity.

If we were to mesh the inside of a pipe with only one cell across the diameter we would essentially doing 1D solution with only one value for velocity etc. If we would use 10 cells there would be 10 values, etc. How many cells is needed cannot be reliably predicted. The used method is essentially to keep refining the mesh until the solution doesn't change anymore.

In our car example we need a sufficient number of cells around all the modeled geometry details. See Fig. 8. Modeling of small details quickly becomes very expensive and often doesn't provide any value to the solution. Therefore we come back to the suitably simplified geometry. As Einstein stated: "*Everything should be made as simple as possible, but not simpler.*" .

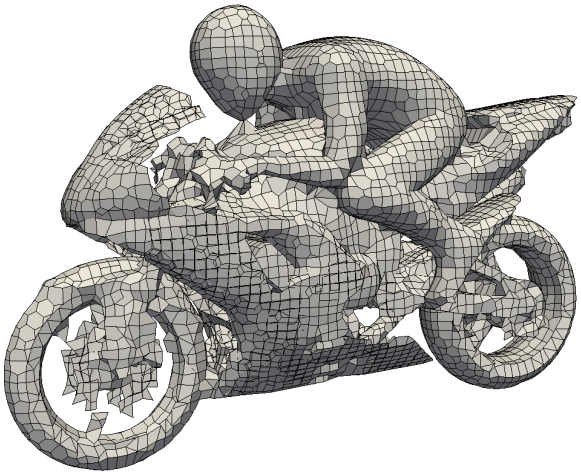


Figure 21: Surface mesh (low quality)

This far we have only paid attention to the mesh near the car surface. As we are interested in the force the air flow causes on the car, the most important phenomena happens near the surface. But we still need to pay attention to the rest of the domain as well.

Great many different meshing strategies exists. The main categories are structured and unstructured meshes. See Fig. 9 for examples.

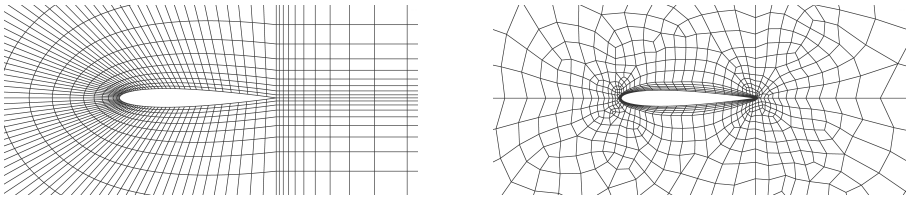


Figure 22: Structured and unstructured (low quality) meshes of a 2D wing

Structural meshes form a block structure where each block is a twisted rectangular block. Unstructured meshes form a more chaotic form. The two strategies can also be combined. In general, structured meshes give better solutions with less cells than unstructured ones. Structured meshes are, however, very difficult to form for complex geometries.

The individual cells can be of any form, but hexahedron, tetrahedron, and polyhedron are the most common ones, see Fig. 3.

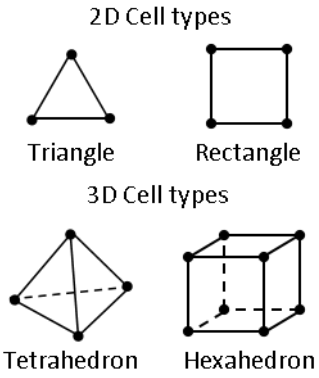


Figure 23: Cell types

2.2.2 Solver

After geometry and mesh have been created it is time to start defining the solver settings. Each solver is a little different but the same basic concepts apply to all.

Governing equations are the equations that define the physics and chemistry inside the calculation domain. Boundary conditions define the borders. The choices for a fluid flow problem would be, for example: compressible or incompressible Navier-Stokes, steady state or unsteady simulation, etc.

Material properties define the behavior of the fluid. We need to choose between constant and varying fluid properties and define them.

Turbulence model is almost always needed in industrial applications. In some lucky cases the flow is laminar and easy to solve but not usually. This course will almost exclusively consider two equation Reynolds averaged turbulence models.

In industrial applications it's not possible to use fine enough mesh and time steps to calculate the small scale fluctuation of the flow. Therefore we use sub models for the small scale phenomena. This modeling, together with mesh quality, is usually the largest source of error in CFD calculations.

At this stage, it's sufficient to say that if you have no better knowledge, always use k-ω-SST model.

Boundary condition define the behavior at the boundary. Its common to define a constant velocity or mass flow for inlet. Temperature etc. also needs boundary conditions.

In industrial problem a considerable problem may be that the boundary conditions are not known. This makes reliable CFD solution near impossible. General understanding about the modeled case helps to identify sensible boundary conditions.

Discretization method means the numerical details of the solution. Discretization is done in space and time. An example of discretization method is linear interpolation in which a unknown value between two known values is assumed to lie on a straight line between the two know values. On the other hand, if there is a strong flow from the direction of a known value to the unknown one, we could assume that the unknown point has the same value as the upwind one. This is called first order upwind scheme.

There exists a great many discretization schemes but most common ones are similar to linear interpolation and upwind scheme.

Time stepping is needed for unsteady calculations. The basic idea is that we know the field values at one time and make some suitable simplification in order to proceed to next time.

Let us consider a rock throwing example for a while. The rock is thrown with some know velocity from some known position at a known time. To numerically solve the rock flight path we could assume for a suitably small amount of time the rock travels in a straight line with constant velocity. We could then calculate a new location for the rock and after calculating the new velocity repeat the process. This would called explicit Euler method.

If we are interested in the unsteady behavior, we need to use a small enough time step to capture all the interesting physics. A too large time step usually also cause the simulation to break.

Iteration method Even if we are solving a steady state problem we usually need to iterate the solution. Most fluid related problems are highly nonlinear and require iteration. Due to the nonlinear nature of the problem, the iteration process is quite sensitive and often breaks.

A typical trick to keep the process converging is to use under relaxation. With under relaxation we don't accept the new solution of out equations as they are, but instead use some value between the old and the new result. This limits the rate of change in the solution and often assists in convergence.

Time stepping and solving of nonlinear equations are remarkably similar and time stepping methods are sometimes used for steady state problems also.

Saving the results is necessary for post processing. It is, however, often inconvenient to save all the results as this would require huge amounts of storage and slow down the solution. One would usually save all field of the final solution and some snapshots of interesting values during the iteration.

2.2.3 Post-processing

Post-processing is the step in which the interesting numbers are extracted from the solution. The solution may consist of hundreds of GB:s of velocity, temperature, etc. data but we only want the mean heat transfer or pressure drag, i.e. one number. Post-processing is also the step in which fancy pictures and videos are made. In addition to being very marketable the figures and videos teach intuitive understanding of the simulation.

For example, below in Fig. 13, we see the temperature and velocity fields at one time step and 2D slice from a 3D solution. The application is a heat ex-

changer. Such colorful pictures are important in marketing. The visualizations are also useful for intuitive understanding.

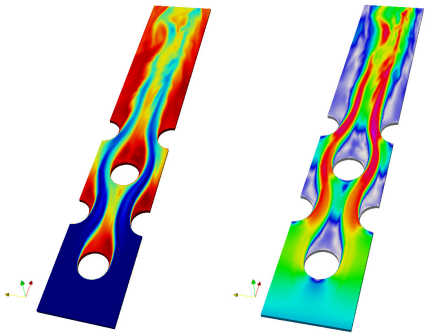


Figure 24: Temperature and velocity fields at one time step[4]

2.3 Convergence

For a steady state case convergence means that the solution of the nonlinear system doesn't chance between iterations.

For a unsteady case convergence may mean that the solution of one time step doesn't change between iterations. It may also mean that the solution has converged to a steady state solution and doesn't change.

To have a meaningful discussion about convergence a convergence criteria is needed. The best convergence criteria are based on the physics of the problem. For example mean heat transfer coefficient of drag force are good convergence criteria.

It is rather typical that a fluid problem doesn't have a steady state solution. In Fig. 14 we see an example for a heat exchanger. The mean heat transfer coefficient (htc) and pressure drop (dp) both start to oscillate around a mean but no convergence occurs.

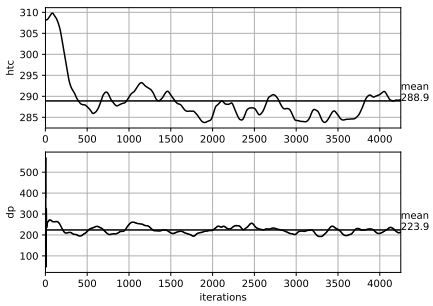


Figure 25: Pseudo convergence of steady state solution

It is usually up to user experience to know when the solution is "good enough". No formal and practical solution exists.

It is common to use a math based criteria called residual for convergence. The residual doesn't usually have any obvious physical meaning and it is near impossible to know when a solution is good enough from the residual. It is better to avoid using them altogether.

2.4 Mesh independency

As stated before, each cell in the mesh only represent a single value in a velocity or temperature field. In for these discrete number to represent the real continuous physics of a fluid flow, a lot of them is needed. Experience is the only reliable way to know how many cells are needed without a formal grid independency test.

A formal grid independency test consists of running a series of simulations on finer and finer meshes. Once the solution doesn't change we can conclude that the solution is grid independent on that mesh topology. Mesh topology means the meshing strategy, cell type, etc. To be totally formal, a series of different topologies would then be needed.

In industry, it is often near impossible to run a formal mesh independency test even for a single mesh topology. The computational cost increases fast with mesh refining. Usually faster than $\mathcal{O}(n^2 \log n)$.

It is, however, often practical to first run a simulation on a coarse mesh and then use the results of that simulation as a initial value on a finer mesh. This may make the calculation faster and gives some understanding of the mesh dependency of the solution.

2.5 Turbulence

2.6 Navier-Stokes equation

2.7 Other equations

3 Heat Conduction (diffusion)

The material discussed in this section may be found in a more elaborated format from Versteeg and Malalasekara 2007, Chapter 5. [5]

In this section we study the general properties of Finite Volume Method (FVM) using one-dimensional heat conduction as an example. The necessary steps are very similar in more complicated cases. For the same information in more elaborated format see the introduction level text book by Versteeg and Malalasekera, An introduction to Computational Fluid Dynamics. For this section see chapter four.

The goal of FVM, and other similar numerical methods, is to divide a continuous problem into small discrete pieces. An approximate solution of the original problem is expressed with a large number of simple arithmetic equations. This results are readily collected 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 and 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 are Finite Element Method (FEM) and Finite Difference Method (FDM). In general, numerical solution of fluid related problems are often 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.).

3.1 Developing the equations for 1D problem

See Versteeg and Malalasekare [5] page 114, Section 4.1-4.3.

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. 26. 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.

We use letter P to refer to the central point (node) of the control volume we are currently interested in and W and E to refer to neighboring nodes as shown in Fig. 26.

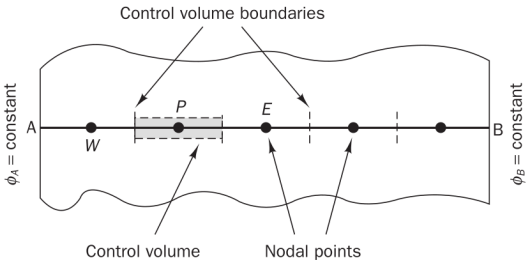


Figure 26: Control volumes and nodes [5]

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. 28. The cell size, or distance from one face to another is called Δx as shown in Fig. 28.

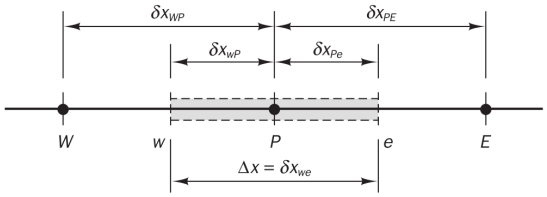


Figure 27: Faces and distances [5]

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 concentrate on one control volume, the one around node P, for a while. If there is a heat flux density $q = -k\frac{\mathrm{d}T}{\mathrm{d}x}$ from neighboring cell W, across the face A_w at the boundary between nodes W and P, to cell P, the heat flux is $(qA)_w = -\left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_w$. We use subscripts for locations, see Fig. 28. 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.

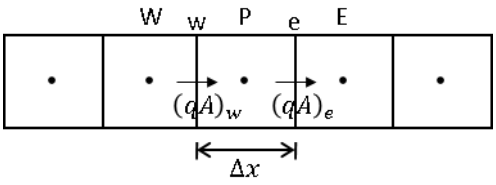


Figure 28: Heat conduction

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

$$
\begin{aligned}
&(qA)_w - (qA)_e + \overline{S}\Delta V = 0 \\
\Leftrightarrow &\left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_e - \left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_w + \overline{S}\Delta V = 0
\end{aligned} \tag{3.1}
$$

The Eq. 3.1 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 discretize 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. If thermal conductivity k and/or cross-section area A varies, they must be evaluated also. Cross-section area A is usually known and requires no special care.

The most common way to discretize gradient term is to use central differencing. The basic idea is to assume linear temperature profile between two nodes as shown in Fig. 29. In essence central differencing is the same thing as linear interpolation.

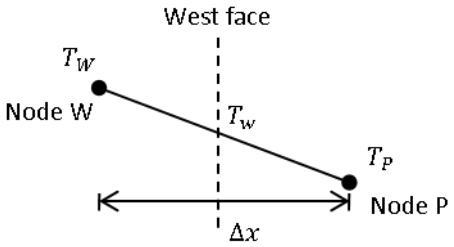


Figure 29: Central differencing

With uniform grid linear interpolation gives at cell faces

$$
\boxed{\begin{aligned} k_w &= \frac{k_W + k_P}{2} \\ k_e &= \frac{k_E + k_P}{2} \end{aligned}} \tag{3.2}
$$

If we have constant thermal conductivity k we can ignore the interpolation in Eq. 3.2. Note that if k at the faces might be known from the problem geometry. In this case there is no need to interpolate even if k varies spacially.

With uniform grid central differencing scheme gives for a temperature gradients

$$
\boxed{\begin{aligned} \left(\frac{\mathrm{d}T}{\mathrm{d}x}\right)_w &\approx \frac{T_P - T_W}{\Delta x} \\ \left(\frac{\mathrm{d}T}{\mathrm{d}x}\right)_e &\approx \frac{T_E - T_P}{\Delta x} \end{aligned}} \tag{3.3}
$$

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

Substituting Eq. 3.3 into Eq. 3.1

$$k_e A_e \frac{T_E - T_P}{\Delta x} - k_w A_w \frac{T_P - T_W}{\Delta x} + \overline{S}\Delta V = 0 \tag{3.4}$$

Now we have a fully discretized equation for heat conduction!
It is often useful to rearrange Eq. 3.4 as

$$\begin{aligned} &\frac{k_e A_e}{\Delta x} T_E - \frac{k_e A_e}{\Delta x} T_P - \frac{k_w A_w}{\Delta x} T_P + \frac{k_w A_w}{\Delta x} T_W + \overline{S}\Delta V = 0 \\ &\Leftrightarrow \underbrace{\left(\frac{k_e A_e}{\Delta x} + \frac{k_w A_w}{\Delta x}\right)}_{a_P = a_W + a_E} T_P = \underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_w} T_W + \underbrace{\left(\frac{k_e A_e}{\Delta x}\right)}_{a_E} T_E + \underbrace{\overline{S}\Delta V}_{S_u} \end{aligned} \tag{3.5}$$

$$\Leftrightarrow \boxed{a_P T_P = a_W T_W + a_E T_E + S_u} \tag{3.6}$$

At this point our equation is in a ready-for-a-computer form but lets make a sanity check first.

From experience 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, lets 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. 3.6 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.

3.1.1 Same thing in math

For one-dimensional steady heat conduction (diffusion) with a volumetric source term we have

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(k\frac{\mathrm{dT}}{dx}\right) + S = 0 \tag{3.7}$$

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

$$\int_{CV} \frac{\mathrm{d}}{\mathrm{d}x}\left(k\frac{\mathrm{dT}}{dx}\right)\mathrm{d}V + \int_{CV} S\mathrm{d}V = 0 \tag{3.8}$$

Using the Gauss divergence theorem we get

$$\begin{aligned} &\int_A \boldsymbol{n}\cdot k\frac{\mathrm{dT}}{dx}\mathrm{d}A + \int_{CV} S\mathrm{d}V = 0 \\ \Leftrightarrow &\left(kA\frac{\mathrm{dT}}{dx}\right)_e - \left(kA\frac{\mathrm{dT}}{dx}\right)_w + \overline{S}\Delta V = 0 \end{aligned} \tag{3.9}$$

With central differencing, see Eqs. 3.2 and 3.3

$$k_e A_e \frac{T_E - T_P}{\Delta x} - k_w A_w \frac{T_P - T_W}{\Delta x} + \overline{S}\Delta V = 0 \tag{3.10}$$

which is the same equations as Eq. 3.4. Rearranging Eq. 3.10 results in Eq. 3.5.

$$\underbrace{\left(\frac{k_e A_e}{\Delta x} + \frac{k_w A_w}{\Delta x}\right)}_{a_P = a_W + a_E} T_P = \underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_w} T_W + \underbrace{\left(\frac{k_e A_e}{\Delta x}\right)}_{a_E} T_E + \underbrace{\overline{S}\Delta V}_{S_u} \tag{3.11}$$

$$\Leftrightarrow \boxed{a_P T_P = a_W T_W + a_E T_E + S_u} \tag{3.12}$$

3.2 Boundary conditions

In Section 3.1 we developed the discretized equations for 1D heat transfer inside the domain. In order to use the equation 3.5 for anything real we now need to give it boundary conditions.

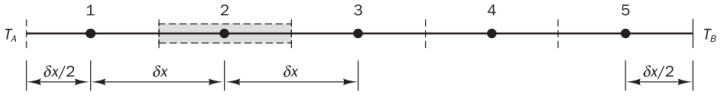


Figure 30: Boundary conditions 1D

The cells that touch boundaries are called boundary cells and require special attention. Let us concentrate on cell 1 in Fig. 30 for a while.

3.2.1 Constant temperature boundary

See Versteeg and Malalasekare [5] 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. 3.2 and 3.3, we get for the west side face in Eq. 3.1

$$\left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_w \approx k_A A_A \frac{T_P - T_A}{\Delta x/2} \tag{3.13}$$

Substituting this in Eq. 3.1 and treating the inside face w in the same way as before results in

$$k_e A_e \frac{T_E - T_P}{\Delta x} - k_A A_A \frac{T_P - T_A}{\Delta x/2} + \overline{S}\Delta V = 0 \tag{3.14}$$

Rearranging Eq. 3.14 gives

$$\underbrace{\left(\frac{k_e A_e}{\Delta x} + \overbrace{\frac{k_A A_A}{\Delta x/2}}^{S_P}\right)}_{a_P = a_E + S_P} T_P = \underbrace{\left(\frac{k_e A_e}{\Delta x}\right)}_{a_E} T_E + \underbrace{\left(\frac{k_A A_A}{\Delta x/2}\right) T_A + \overline{S}\Delta V}_{S_u}$$
$$\Leftrightarrow a_P T_P = a_E T_E + S_u \tag{3.15}$$

Comparing Eqs. 3.5 and 3.15 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. 3.3.

If the constant temperature is set on the B boundary as shown in Fig. 30 we follow a similar procedure as for A boundary. Now we replace East side coefficient a_E with suitable source terms and arrive to

$$\begin{gathered}\underbrace{\left(\frac{k_w A_w}{\Delta x} + \overbrace{\frac{k_B A_B}{\Delta x/2}}^{S_P}\right)}_{a_P = a_W + S_P} T_P = \underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_W} T_W + \underbrace{\left(\frac{k_B A_B}{\Delta x/2}\right) T_B + \overline{S}\Delta V}_{S_u} \\ \Leftrightarrow a_P T_P = a_W T_W + S_u\end{gathered} \tag{3.16}$$

3.2.2 Zero gradient (insulated) boundary

See Versteeg and Malalasekare [5] page 125, Example 4.3.

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

$$q = -k \underbrace{\frac{\mathrm{d}T}{\mathrm{d}x}}_{0} = 0 \tag{3.17}$$

To apply the zero gradient boundary condition for East face in Eq. 3.1 we but the east side coefficient to zero as $a_E = 0$ and from Eq. 3.5 we get

$$\begin{gathered}\underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_P = a_W} T_P = \underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_w} T_W + \underbrace{\overline{S}\Delta V}_{S_u} \\ \Leftrightarrow a_P T_P = a_W T_W + S_u\end{gathered} \tag{3.18}$$

No additional source terms are needed.

3.3 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 equations form

$$\begin{aligned}\boldsymbol{AT} &= \boldsymbol{b} \\ \Leftrightarrow \boldsymbol{T} &= \boldsymbol{A^{-1}b}\end{aligned} \tag{3.19}$$

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 31: 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. $\frac{\mathrm{dT}}{\mathrm{d}x} = 0$. 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 uniform mesh. We have uniform volumetric heat generation S.

For the inside cells, dividing by the constant area A we have from Eq. 3.5

$$\begin{gathered}\underbrace{\left(\frac{k}{\Delta x}+\frac{k}{\Delta x}\right)}_{a_P=a_W+a_E}T_P=\underbrace{\left(\frac{k}{\Delta x}\right)}_{a_w}T_W+\underbrace{\left(\frac{k}{\Delta x}\right)}_{a_E}T_E+\underbrace{S\Delta x}_{S_u} \\ \Leftrightarrow a_P T_P = a_W T_W + a_E T_E + S_u\end{gathered} \tag{3.20}$$

For left boundary, dividing by the constant area A we have from Eq. 3.15

$$\begin{gathered}\underbrace{\left(\frac{k}{\Delta x}+\overbrace{\frac{k}{\Delta x/2}}^{S_P}\right)}_{a_P=a_E+S_P}T_P=\underbrace{\left(\frac{k}{\Delta x}\right)}_{a_E}T_E+\underbrace{\left(\frac{k}{\Delta x/2}\right)T_A+\overline{S}\Delta V}_{S_u} \\ \Leftrightarrow a_P T_P = a_E T_E + S_u\end{gathered} \tag{3.21}$$

For right boundary, dividing by the constant area A we have from Eq. 3.18

$$\begin{gathered}\underbrace{\left(\frac{k}{\Delta x}\right)}_{a_P=a_W}T_P=\underbrace{\left(\frac{k}{\Delta x}\right)}_{a_w}T_W+\underbrace{\overline{S}\Delta V}_{S_u} \\ \Leftrightarrow a_P T_P = a_W T_W + S_u\end{gathered} \tag{3.22}$$

We may now observe from Eqs. 3.20-3.22 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

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -a_W & a_W & 0 & 0 & 0 \\ 0 & -a_W & a_W & 0 & 0 \\ 0 & 0 & -a_W & a_W & 0 \\ 0 & 0 & 0 & -a_W & a_W \end{bmatrix} \tag{3.23}$$

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. 3.23 gives

$$\boldsymbol{A} = \begin{bmatrix} a_E & -a_E & 0 & 0 & 0 \\ -a_W & a_W + a_E & -a_E & 0 & 0 \\ 0 & -a_W & a_W + a_E & -a_E & 0 \\ 0 & 0 & -a_W & a_W + a_E & -a_E \\ 0 & 0 & 0 & -a_W & a_W \end{bmatrix} \tag{3.24}$$

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. 3.21 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

$$\boldsymbol{A} = \begin{bmatrix} a_E + \frac{k}{\Delta x/2} & -a_E & 0 & 0 & 0 \\ -a_W & a_W + a_E & -a_E & 0 & 0 \\ 0 & -a_W & a_W + a_E & -a_E & 0 \\ 0 & 0 & -a_W & a_W + a_E & -a_E \\ 0 & 0 & 0 & -a_W & a_W \end{bmatrix} \tag{3.25}$$

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

$$\boldsymbol{b} = \begin{bmatrix} \left(\frac{k}{\Delta x/2}\right) T_B & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.26}$$

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 final step we need to add the volumetric source to our source vector $\boldsymbol{b}$. The volumetric source is the same in all Eqs. 3.20-3.22, $S_u = S\Delta x$. By adding it to the source vector we get

$$\boldsymbol{b} = \begin{bmatrix} \left(\frac{k}{\Delta x/2}\right) T_B + S\Delta x & S\Delta x & S\Delta x & S\Delta x & S\Delta x \end{bmatrix}^T \tag{3.27}$$

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. 31. 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()
```

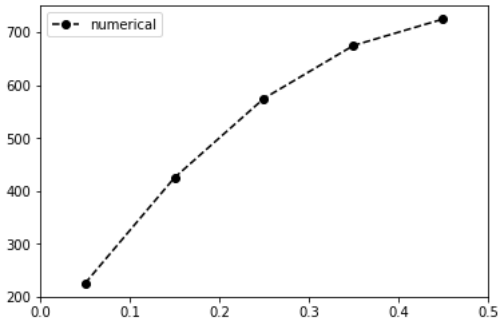


Figure 32: Plotting

3.4 Unsteady

See Versteeg and Malalasekare [5] page 243, Chapter 8.

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

$$\rho c \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right) + S \tag{3.28}$$

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 [5].*

Let us first consider the left hand side of Eq. 3.28, $\rho c \frac{\partial T}{\partial t}$. We may discretized it as

$$\rho c \frac{\partial T}{\partial t} \approx \rho c \frac{T - T^0}{\Delta t} \tag{3.29}$$

where Δt is time step and T^0 is temperature from previous time step. If we now substitute Eq. 3.29 into Eq. 3.28 we get

$$\rho c \frac{T - T^0}{\Delta t} = \frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right) + S \tag{3.30}$$

Using FVM to the left hand side of Eq. 3.30

$$\int_{CV} \rho c \frac{T - T^0}{\Delta t} dV \approx \rho c A \frac{T - T^0}{\Delta t} \Delta x \tag{3.31}$$

We may now recognize the right hand side of Eq. 3.31, $\frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right) + S$ as the governing equation for steady state heat conduction, Eq. 3.7. 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. 3.30. 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. We choose to use the new time. This is caller 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. 3.30, 3.31, 3.5.

$$\begin{aligned}
&\underbrace{\rho c A \frac{\Delta x}{\Delta t}}_{a_P^0}(T_P - T_P^0) + \underbrace{\left(\frac{k_e A_e}{\Delta x} + \frac{k_w A_w}{\Delta x}\right)}_{a_W + a_E} T_P = \underbrace{\left(\frac{k_w A_w}{\Delta x}\right)}_{a_w} T_W + \underbrace{\left(\frac{k_e A_e}{\Delta x}\right)}_{a_E} T_E + \underbrace{\overline{S}\Delta V}_{S_u} \\
&\Leftrightarrow a_P^0 T_P - a_P^0 T_P^0 + a_W T_P + a_E T_P = a_w T_W + a_E T_E + S_u \\
&\Leftrightarrow \boxed{\underbrace{(a_P^0 + a_W + a_E)}_{a_P} T_P = a_w T_W + a_E T_E + a_P^0 T_P^0 + S_u}
\end{aligned} \tag{3.32}$$

From Eq. 3.32 we see that the solution of one time step in unsteady case is very similar to the steady one. To progress the solution repeat the process in a loop. An example is given in Fig. 33.

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

Figure 33: Time stepping

It is often convenient to collect the time-independent terms into a separate source vector, see *bConstant* in Fig. 33. This often results in simpler and faster code.

3.5 2D and 3D cases

See Versteeg and Malalasekare [5] 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. 34.

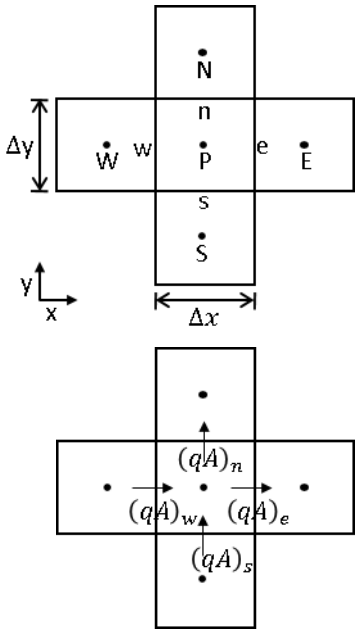


Figure 34: 2D notations and control volume

Forming a balance equation for 2D heat conduction results in

$$\begin{aligned} &(qA)_w - (qA)_e + (qA)_s - (qA)_n + \overline{S}\Delta V = 0 \\ \Leftrightarrow &\left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_e - \left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_w + \left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_n - \left(kA\frac{\mathrm{d}T}{\mathrm{d}x}\right)_s + \overline{S}\Delta V = 0 \end{aligned} \quad (3.33)$$

With finite volume method, using central differencing and linear interpolation Eq. 3.33 gives

$$\begin{aligned} \underbrace{\left(\frac{k_eA_e}{\Delta x} + \frac{k_wA_w}{\Delta x} + \frac{k_sA_s}{\Delta y} + \frac{k_nA_n}{\Delta y}\right)}_{a_P = a_W + a_E + a_S + a_N} T_P = &\underbrace{\left(\frac{k_wA_w}{\Delta x}\right)}_{a_w} T_W + \underbrace{\left(\frac{k_eA_e}{\Delta x}\right)}_{a_E} T_E \\ &+ \underbrace{\left(\frac{k_sA_s}{\Delta y}\right)}_{a_S} T_S + \underbrace{\left(\frac{k_nA_n}{\Delta y}\right)}_{a_N} T_N \\ &+ \underbrace{\overline{S}\Delta V}_{S_u} \end{aligned} \quad (3.34)$$

$$\Leftrightarrow \boxed{a_PT_P = a_WT_W + a_ET_E + a_ST_S + a_NT_N + S_u} \quad (3.35)$$

As we can see from Eqs. 3.34-3.35 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 deriving the Eq. 3.34 by yourself and programming your own 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 3.7.1.

3.6 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

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right)+S=0 \tag{3.36}$$

it is customary to use either a vector or tensor notation instead. With vector notation Eq. 3.36 becomes

$$\nabla\cdot k\nabla T+S=0 \tag{3.37}$$

and with tensor notation

$$\frac{\partial}{\partial x_j}\left(k\frac{\partial T}{\partial x_j}\right)+S=0 \tag{3.38}$$

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.

3.7 Extra material

Extra material. Not needed on this course.

3.7.1 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. 35. See https://docs.scipy.org/doc/scipy/reference/sparse.html for more details.

```python
import scipy.sparse as sparse
from scipy.sparse import linalg as splinalg

class SparseMatrixA(object):
    def __init__(self):
        self.row = []
        self.col = []
        self.val = []

    def add(self, row, col, val):
        self.row.append(row)
        self.col.append(col)
        self.val.append(val)

    def finalize(self):
        return sparse.csr_matrix(sparse.coo_matrix(
                                 (self.val, [self.row, self.col])
                                ))

    def solve(self,b, A=None):
        if A is None:
            A = self.finalize()
        return splinalg.spsolve(A, b)
```

Figure 35: 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.

3.7.2 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)

$$\nabla \cdot \sigma \nabla \phi = 0 \tag{3.39}$$

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

$$\boldsymbol{E} = -\nabla \phi \tag{3.40}$$

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

$$\boldsymbol{J} = \sigma \boldsymbol{E} \tag{3.41}$$

and finally the volumetric electric heating power p

$$p = \frac{\partial P}{\partial V} = \boldsymbol{J} \cdot \boldsymbol{E} = \boldsymbol{J} \cdot \boldsymbol{J}/\boldsymbol{\sigma} = \frac{\mid \boldsymbol{J} \mid^{\mathbf{2}}}{\boldsymbol{\sigma}} \tag{3.42}$$

Electric conductivity can be calculated from electric resistivity ρ as

$$\sigma = \frac{1}{\rho} \tag{3.43}$$

Linear Mechanics For small deformations

$$-\nabla \cdot \boldsymbol{\sigma} = \boldsymbol{f} \tag{3.44}$$

$$\boldsymbol{\sigma} = \lambda \nabla \cdot \boldsymbol{u}\mathbf{I} + 2\mu \boldsymbol{D} \tag{3.45}$$

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. 3.44.

Part I
TODO

4 Advection-diffusion

5 Navier-Stokes

5.1 The NS equation

5.2 Compressible vs. incompressible

5.3 Pressure coupling

5.4 Boundary conditions

6 Mesh

7 Turbulence

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 ϕ

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

For a vector $\boldsymbol{V}$

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

A.2.2 Divergence

For a vector $\boldsymbol{V}$

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

A.2.3 Laplacian

For a scalar ϕ

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

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] fkevin. semi truck desert. [Online; accessed 15-Jan-2018] CC0.

[3] fkevin. Simplified truck geometry for cfd analysis, 2015. [Online; accessed 15-Jan-2018].

[4] Turo Valikangas, 2018. Personal commmunication.

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

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