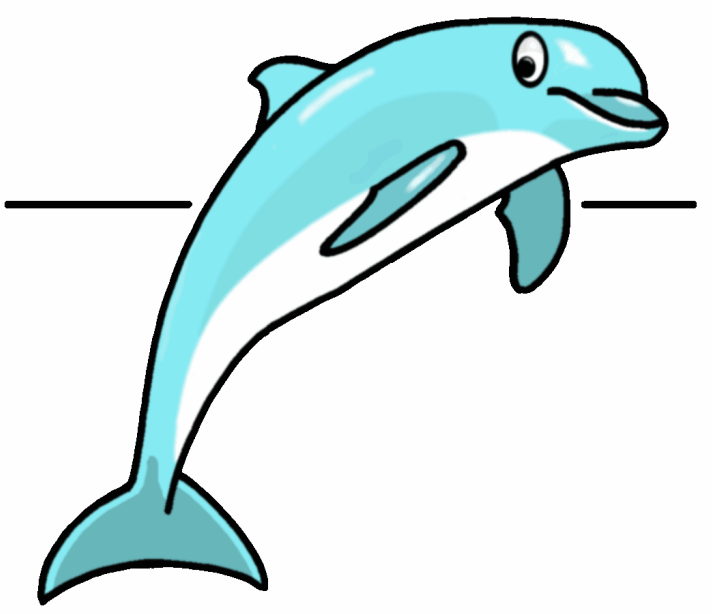
CYCLONE
Fluid Dynamics

Dolfyn Developers Guide

DRAFT



CFD-08xxxx
15th September 2008

Cyclone Fluid Dynamics B.V.
Sweelincklaan 4 Tel.: +31-40-22 30 491 Web: www.cyclone.nl
NL–5583 XM Waalre Fax.: +31-40-22 30 490 email: info@cyclone.nl

CYCLONE
Fluid Dynamics

Dolfyn Developers Guide

DRAFT

Cyclone Fluid Dynamics BV

Author:
H.W. Krüs

CFD-08xxxx
Cyclone Fluid Dynamics B.V.
15th September 2008

Cyclone Fluid Dynamics B.V.
Sweelincklaan 4 Tel.: +31-40-22 30 491 Web: www.cyclone.nl
NL–5583 XM Waalre Fax.: +31-40-22 30 490 email: info@cyclone.nl

Preface

This is just a collection of notes, thoughts and ideas about the dolfyn code. Do not expect a decent structure nor an in depth methodology manual. Consult for the latter the various appropriate text books.

Thanks to Yousef Saad of University of Minnesota, Department of Computer Science and Engineering for Sparsekit2 (www-users.cs.umn.edu/∼saad/) and the people behind Lapack (www.netlib.org/lapack/) and countless others like those involved in the Hypre and VisIt packages at Lawrence Livermore (www.llnl.gov/CASC/linear_solvers/ respectively www.llnl.gov/visit/)

A special thanks goes to Franco Magagnato of the Department of Fluid Machinery, University of Karlsruhe, for sharing his Sparc code years ago (Structured PArallel Research Code). Sparc showed and proved the concept of f95 and MPICH on a heterogeneous network consisting of different machines and operating systems. Within a day the code was up and running over a network with Alpha/Unix and PC/Linux workstations. A quality statement for the three pieces of software involved (Sparc, f95 and MPICH).

Thanks also to those involved in for example TEX, LATEX and pdfTEX, xfig, NEdit, the Gimp, Perl, Gnu and Linux. Intel made it's contribution by releasing their Intel Fortran95 compiler (ifc/ifort) for free to the Linux community. And of course, Andy Vaught for his amazing work on g95 (www.g95.org). Thanks to IBM for Data Explorer or OpenDX as it is now called (www.opendx.org). An impressive tool. Also CiteSeer at Penn State was, and is, very useful.

Simplicity and accuracy are the keywords. This means for example that there are no 2D and axisymmetric modes of the code. Also readability is more important than speed (rely instead on the quality of today's compilers and on the array syntax of f95).

Dolfyn is based on the ideas, algorithms, and procedures presented in the book by Joel Ferziger & Milovan Perić 'Computational Methods for Fluid Dynamics' [12]. This book is also acts as Theoretical Manual.

Dolfyn contains contributions by Thomas Geenen, Johan Jacobs, Shibo Kuang, Max Staufer, Bouke Tuinstra.

Enjoy!

Contents

List of Figures

The purpose of computing is *insight*, not numbers.
R. Hamming, 1962

1 Introduction

Heat and mass transfer, transport phenomena, and fluid dynamics play an important role in various products, machines and in our daily life.

Examples are:

- Automotive industry
  - External flow around vehicles
  - Fouling
  - Thermal passenger comfort
- Heating and cooling
  - Heat exchangers
  - Cooling of electronics
- Flows in pipes and tubes
  - Industrial installations
  - Air conditioning
- Wind in the built environment
  - Wind forces
  - Wind nuisance for pedestrians and cyclists
- Process technology

dolfyn is a face based implicit Finite Volume Method code, employing primitive variables on 3D unstructured polyhedral meshes targeted towards these industrial type of problems.

dolfyn is accompanied by a preprocessor. The preprocessor writes a geometry file in a format suitable for dolfyn. The input is a set of three files which describe the cells, vertices en the boundaries (see Appendix A). Dolfyn reads the geometry file and a separate input file (format is described in a separate document). In this input file the user sets numerical and model data, boundary conditions etc. The input file can be edited with any simple editor. The file structure is shown in Figure 1.1.

The code uses a ‘segregated solver’ which means that the transport equations are solved sequentially. Because the coupled non-linear equations have been

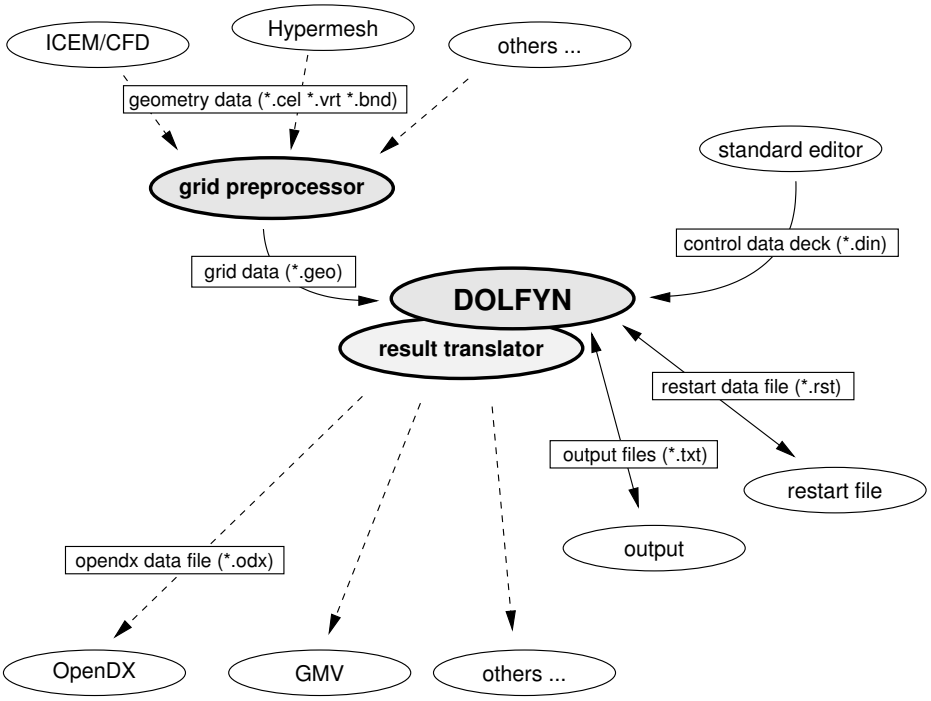


Figure 1.1: Dolfyn data and result files

linearised implicitly, several iterations are needed in order to get a converged solution. Using the currently known fluid properties and mass fluxes the three momentum transport equations are solved. At this stage the continuity equation may not be satisfied and a pressure correction is set up. This correction is solved to obtain the correct pressure and velocity fields and the mass fluxes at the faces satisfy the continuity equation. After this stage the transport equations for the turbulence models, energy or species is solved. Finally the fluid properties are adjusted and the process is repeated until convergence.

The grid or mesh used by dolfyn is based on ‘face based unstructured polyhedral cells’. Unstructured does not mean tetrahedral cells only (in 3D, or triangles in 2D), but refers to the way the topology is implemented. In using polyhedra any type of mesh can be used, even ‘structured hexahedra’s’. It only depends on the kind of preprocessor, or ‘mesher’, or ‘grid generator’ one is using. The same applies to the postprocessor to visualise the final results; a postprocessor which only can handle hexahedral cells is of no use when one employs tetrahedral cells. On the other hand results of hexahedral cells can be converted to a tetrahedral structure and displayed (this technique is partly used for OpenDX for example). However, ‘real polyhedra’ still pose a problem (except, maybe, for GMV, but GMV is not Open Source). Thus dolfyn is never the limiting code in the process and it all depends on the kind of, favourite, pre- and postprocessors at hand.

1.1 Fortran 9x

Dolfyn uses Fortran? But isn't that hopelessly old fashioned?

Lots of people who have not encountered the latest Fortran compilers react in this way. Fortran 77 is indeed a bit 'old fashioned' and had some serious limitations (however not it's speed).

The language was developed way back in 1954 at IBM and has been updated since. Well known versions are Fortran IV (or Fortran 66), Fortran 77 (the well known old standard), Fortran 90 and soon followed by Fortran 95. In 2003 further enhancements were made, notably enhancements of derived types, asynchronous transfer, stream access and procedure pointers and, very important, much improved interoperability with the C programming language. If number crunching is what you are up to then Fortran (95+) still is the best horse for your course. With g95 and f03gl you can even do OpenGL!

Modern Fortran is also highly portable. It is relatively easy to write an entirely portable program in Fortran, even without recourse to a preprocessor. Nor does one needs tools like the configure and build system ('automake', 'autoconf', and 'configure' etcetera). Hunting down 'include files' all over the place and file name case sensivity are absent (something one has to take care of when developing a multiplatform solution). And finally, not to mention Fortran's vectorisation and parallelisation capalilities are excellent.

If the before mentioned arguments are not enough. Well, the most important argument for Fortran is it's readability. Even people who hardly know the language can read and understand the code (see the example in Figure 1.2). "Another important point is that the time required to learn Fortran 90 is much less than the time to learn either C or C++. ... But the issue is what freshmen should learn as their first language and for that I would recommend Fortran 90 hands down."[1] Readability should not be underestimated since it is the core of the peer-reviewed human eye parallel debugging device.

```
.
.
Xn = Face(i)%n                  ! surface vector
call normalise(Xn)              ! normal vector with length 1

Up = Up - Uw                    ! velocity difference vector

dp = dot_product( Up , Xn )     ! dot product of the two vectors
Un = dp * Xn                    ! normal velocity component
Ut = Up - Un                    ! tangential velocity component
.
.
```

Figure 1.2: Example Fortran95 code

[1] "Fortran 90 vs C++ - An educational perspective", by Dr. John K Prentice, Quetzal Computational Associates, 1996. Note that Fortran 90 is addressed and advocated, imagine Dr. Prentice's enthusiasm about Fortran 95 or even Fortran 2003.

The current July 2008 version of dolfyn only consists of about 25,000 lines of Fortran code. About 60% of it, in the order of 15,000 lines of code, is actually related to dolfyn itself. 16% of the lines only deal with reading the din control file and setting the control parameters (easy of use always tends to increase the 'interactive part' of a code). Another 10% are subroutines for the various output interfaces. Finally 18% accounts for the Fortran77 SparseKit2 and Lapack subroutines.

1.2 Further reading

Of course the basics of dolfyn can be found in Ferziger and Perić' 'Computational Methods for Fluid Dynamics' [11][12]. Older classic textbooks are by Roache [34] and Patankar [29], the influence of the latter is not to be underestimated. Today 'Patankar' is useful as an introduction but 'proves' that a co-located code, like dolfyn, is not feasible. The breakthrough came with the Rhie and Chow paper [32] in 1983. But 'staggered grids' still have their supporters like Wesseling [46][44].

Other interesting introductions are Anderson's 'Computational Fluid Dynamics: The Basics with Application' [9], however it is more geared towards aerospace applications.

Dolfyn relies heavily on the work and theses by people like Spalding, Gosman, Issa, Jasak, Weller and many others at Imperial College (examples are [16][30][24]). Imperial College is at the roots of various leading commercial CFD codes (from the early eighties onwards).

Another stronghold is the T-3 group at Los Alamos [26] were k-ε was born in 1967 [18][19][17] and the Volume-Of-Fluid, VOF, method in the eighties. Los Alamos has a long tradition in open source codes (the Marker-and-Cell, MAC, method by Harlow and Welch, the first successful code with primitive variables thanks to a novel staggered scheme [13][20][43][5]. Followed by 'SOLA' [23], 'SOLA-VOF' by Hirt and Nichols [28], ALE [22] and other codes, and the impressive 'Kiva' series of combustion codes by Hirt, Amsden and others [33][8][7][14][6][3][4].

More on boundary layers can be found in Schlichting [36], and on turbulence start with Tennekes and Lumley [39] and then Hinze [21]. Turbulence modelling for CFD is dealt with by Wilcox [45]. Another book with much attention to modelling is Pope's 'Turbulent flows' [31].

On general working with CFD one can use Versteeg and Malalasekera's 'An Introduction to Computational Fluid Dynamics' (use the latest 2007 edition [42] instead of the 1995 edition [41]). Shaw's 'Using Computational Fluid Dynamics' [37] is even older (1992) and is today not so useful anymore.

Do not start with the numerics without proper background in fluid dynamics. Consult your local popular textbooks. And keep Van Dyke's 'Album of fluid

motion' [40] and other books like 'Visualized Flow' by Nakayama [27] at hand.

2 A transported scalar

The basic ideas of how a transport equation can be solved will be illustrated in this chapter.

The basis is formed by the assumption that a variable varies within a control volume according to

$$\phi(\vec{x}) = \phi_P + (\vec{x} - \vec{x}_P) \cdot (\nabla\phi)_P. \tag{2.1}$$

With the centroid $\vec{x}_P$ defined as:

$$\int_{V_P} (\vec{x} - \vec{x}_P)\, dV = 0. \tag{2.2}$$

Now when equation (2.1) is integrated over volume V results in

$$\begin{aligned} \int_{V_P} \phi(\vec{x})\, dV &= \int_{V_P} \phi_P + (\vec{x} - \vec{x}_P) \cdot (\nabla\phi)_P\, dV \\ &= \phi_P \int_{V_P} dV + [\underbrace{\int_{V_P} (\vec{x} - \vec{x}_P)\, dV}_{=0}] \cdot (\nabla\phi)_P \\ &= \phi_P V_P \end{aligned} \tag{2.3}$$

This means that equation (2.2) is an essential geometrical proposition. Because $\vec{x}_P$ is in the centre and the variable varies linearly; the value of ϕ at P is equal to the mean value, and therefore $\int_V \phi(\vec{x})dV$ is equal to $\phi_P V_P$ with second order accuracy.

Imagine a known flow field (which might be at rest as well) in which the scalar quantity ϕ like a temperature or a concentration is transported. For some *control volume* one can write down the following conservation equation:

$$\underbrace{\frac{\partial}{\partial t}(\rho\,\phi)}_{\text{unsteady term}} + \underbrace{\text{div}(\,\rho\,\vec{v}_r\,\phi\,)}_{\text{convective term}} = \underbrace{\text{div}(\,\Gamma_\phi\,\text{grad}\,\phi\,)}_{\text{diffusive term}} + \underbrace{q_\phi}_{\text{source term}}. \tag{2.4}$$

Where $\vec{v}_r$ is the relative velocity $(\vec{v} - \vec{v}_c)$ between the fluid moving with $\vec{v}$ and the sweeping coordinate velocity $\vec{v}_c$.

It simply states that the change in time of ϕ is the result of the balance between the transport (convection) and diffusion in and out of the the control volume, and, when present, the result of a source (or sink).

Equation (2.4) for an arbitrary moving volume V with surface S transforms into

$$\frac{d}{dt}\int_V \rho\,\phi\, dV + \int_S (\rho\,\vec{v}_r\,\phi) \cdot d\vec{S} = \int_S (\,\Gamma_\phi\,\text{grad}\,\phi) \cdot d\vec{S} + \int_V q_\phi\, dV \tag{2.5}$$

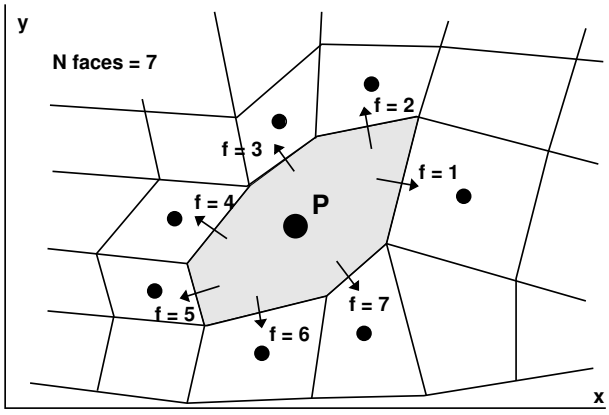


Figure 2.1: An arbitrary control volume

where $\vec{S}$ is the surface vector and the relative velocity is now related to the moving surface.

For steady state cases without sources (or sinks) this equation reduces to a simple *convection/diffusion equation*:

$$\mathrm{div}(\ \rho\ \vec{v}\ \phi\ ) - \mathrm{div}(\ \Gamma_\phi\ \mathrm{grad}\ \phi\ ) = 0 \tag{2.6}$$

or using Gauss' theorem

$$\int_S \rho\phi\vec{v}\cdot\vec{n}\ dS - \int_S \Gamma\ \mathrm{grad}\ \phi\cdot\vec{n}\ dS = 0 \tag{2.7}$$

and can be casted into:

$$\sum_{f=1}^{\mathrm{Nfaces}} F_f^c - \sum_{f=1}^{\mathrm{Nfaces}} F_f^d = 0$$

or

$$\sum_{i=f}^{\mathrm{Nfaces}} (F_f^c - F_f^d) = 0$$

with

$$F_f^c = \int_S \rho\phi\vec{v}\cdot\vec{n}\ dS \approx \dot{m}_f\phi_f \tag{2.8}$$

and $\dot{m}_f$ as the mass flux through face 'f':

$$\dot{m}_f = \int_{S_f} \rho\vec{v}\cdot\vec{n}\ dS \approx (\rho\vec{v}\cdot\vec{n})_f\ S_f \tag{2.9}$$

As the face can be arbitrarily oriented all the velocity components can contribute to the mass flux. Thus each velocity component is multiplied by the corresponding surface vector component (see 8.6.1).

$$\dot{m}_f = \rho_f(S^x u_x + S^y u_y + S^z u_z)_f \tag{2.10}$$

Now assume that the fluid properties ρ and Γ are known (eg. from a previous iteration). Then the value of ϕ and its gradient normal to the cell face is needed.

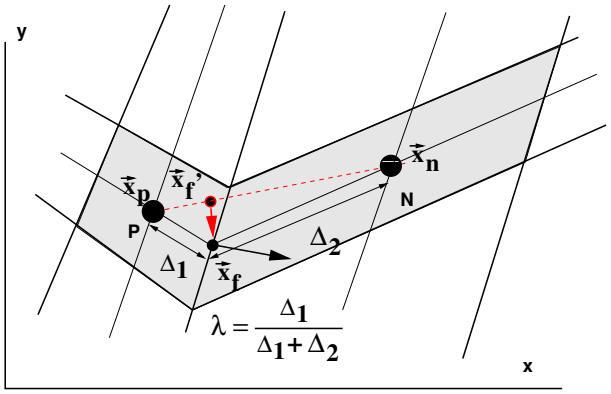


Figure 2.2: Correction

Recall that the normal of a cell face is positive directed outwards (in Dolfyn defined by the Face(i)%cell1 and Face(i)%cell2, the first one is by definition node 'p' and the second, if present, the 'next' node 'n').

The values at the faces are needed at the face centre. The second order linear interpolation of two ϕ's is on a equidistant grid:

$$\phi_f = \frac{\phi_P + \phi_N}{2}$$

and for non-equidistant grids:

$$\phi_f = \lambda_f \phi_N + (1 - \lambda_f)\phi_P \tag{2.11}$$

with λ defined as:

$$\lambda_f = \frac{x_f - x_P}{x_N - x_P} \tag{2.12}$$

The factor λ can be calculated and stored in advance. This is the 'central differencing scheme' (CDS).

For non-smooth grids however the before mentioned accuracy will probably be lost. The value is not evaluated at $\vec{x}_f$ but at $\vec{x}_{f'}$ which is defined by the ratio of the original lengths (λ) and lies on the straight line connecting the two centres. Part of the accuracy can be restored if one evaluates the gradient of ϕ at $\vec{x}_{f'}$ $(= (\nabla\phi)_{f'})$ and travels from $\vec{x}_{f'}$ to $\vec{x}_f$. This is shown in Figure 2.2.

For equidistant grids, and some smooth expanding grids, the correction is not necessary ($\vec{x}_{f'} - \vec{x}_f = 0$). For highly distorted grids the procedure might be repeated several times (hence the variable Ngrad in subroutine GradientPhiGauss).

There is also a simpler first order method called 'upwind differencing scheme' (UDS), in which the upstream nodal value is used:

$$\dot{m}_f = \begin{cases} \phi_P & \text{if} \quad (\vec{v}\cdot\vec{n})_f > 0 \\ \phi_N & \text{if} \quad (\vec{v}\cdot\vec{n})_f < 0 \end{cases} \tag{2.13}$$

The upwind differencing scheme is robust and never results in under- or overshoots, nor is it oscillatory. However, it is inaccurate especially when the flow is

oblique trough the cell face. Thus it is possible to combine both scheme's into a blended or hybrid scheme:

$$F_f = (1-\beta)\, F_f^{\text{uds}} + \beta\, F_f^{\text{cds}} \tag{2.14}$$

or the using a *deffered correction* approach

$$F_f = \underbrace{F_f^{\text{uds}}}_{\text{implicit}} + \underbrace{\gamma \left(F_f^{\text{cds}} + F_f^{\text{uds}}\right)^{\text{old}}}_{\text{explicit}} \tag{2.15}$$

a hybrid scheme in which the higher order scheme is evaluated explicitely as a corrector step to the stable lower order implicit scheme.

Thus the flux can be approximated by:

$$F_f^c = \begin{cases} \max(\dot{m}_f, 0)\, \phi_P + \min(\dot{m}_f, 0)\, \phi_N & \text{for UDS} \\ \dot{m}_f(1-\lambda_f)\, \phi_P + \dot{m}_f \lambda_f\, \phi_N & \text{for CDS} \end{cases} \tag{2.16}$$

For the diffusive fluxes one needs the gradient vector at the cell face centre. The CDS scheme is normally used:

$$F_f^d = \int_{S_f} \Gamma \,\text{grad}\, \phi \cdot \vec{n}\, dS \approx (\Gamma \text{grad}\, \phi \cdot \vec{n})_f\, S_f \tag{2.17}$$

For simple grids one can simply use:

$$\left(\frac{\partial \phi}{\partial n}\right)_{face} = (\nabla \phi)_{face} \cdot \vec{x}_{face} \approx \frac{\phi_N - \phi_P}{|\vec{x}_N - \vec{x}_P|}$$

As both values at P and N show up; this relationship can be used simply in an implicit form:

However this form is too simple for complex meshes. A correction term is discussed in paragraph XXX.

So for one control volume there are a number of fluxes, each of them related between the central node 'P' and the neighbour defined by the face.

surface area components s_x, s_y and s_z ==> normal

2.1 Conservation of momentum

Consider the conservation equations for mass and momentum in the form of equation (2.4) for an arbitrary moving volume V with surface S:

$$\frac{d}{dt}\int_V \rho\, dV + \int_S (\rho\, \vec{v}_r) \cdot d\vec{S} = 0$$

and

$$\frac{d}{dt}\int_V \rho u_i\, dV + \int_S (\rho u_i \vec{v}_r) \cdot d\vec{S} = \int_S \vec{T} \cdot d\vec{S} + \int_V \rho b_i\, dV$$

With b_i the 'body force' in the direction of the Cartesian coordinate x_i (for example gravity, centrifugal and Coriolis forces, electromagnetic forces etc.).

The stress tensor $\vec{T}$ is the molecular rate of transport of momentum and incorporates the surface forces (pressure, normal and shear stresses, surface tension etc.).

Introducing the outwards pointing unit normal $\vec{n}$ to surface $d\vec{S} = \vec{n}dS$ results in:

$$\frac{d}{dt}\int_V \rho \, dV + \int_S (\rho \, \vec{v}_r) \cdot \vec{n} \, dS = 0 \tag{2.18}$$

and

$$\frac{d}{dt}\int_V \rho u_i \, dV + \int_S (\rho u_i \vec{v}_r) \cdot \vec{n} \, dS = \int_S \vec{t_i} \cdot \vec{n} \, dS + \int_V \rho b_i \, dV \tag{2.19}$$

For isotropic Newtonian fluids the components τ_{ij}, of the viscous stress tensor are:

$$\tau_{ij} = -p\,\delta_{ij} + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\,\delta_{ij}\,\mu\,\mathrm{div}\vec{v} \tag{2.20}$$

with the dynamic fluid viscosity μ and the Kronecker delta δ_{ij}.

The stress tensor

$$\Pi = \begin{pmatrix} \tau_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \tau_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \tau_{zz} \end{pmatrix}$$

with $\tau_{xx} = \sigma_x$ etc, and $\tau_{xy} = \tau_{yx}$, $\tau_{xz} = \tau_{zx}$ and $\tau_{yz} = \tau_{zy}$ only contains six elements and is symmetric

$$\Pi = \begin{pmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{pmatrix} \tag{2.21}$$

and the elements are

$$\begin{aligned} \sigma_x &= -p - \tfrac{2}{3}\mu \ \mathrm{div}\ \vec{v} + \mu(\tfrac{\partial u}{\partial x} + \tfrac{\partial u}{\partial x}) \\ \sigma_y &= -p - \tfrac{2}{3}\mu \ \mathrm{div}\ \vec{v} + \mu(\tfrac{\partial v}{\partial y} + \tfrac{\partial v}{\partial y}) \\ \sigma_z &= -p - \tfrac{2}{3}\mu \ \mathrm{div}\ \vec{v} + \mu(\tfrac{\partial w}{\partial z} + \tfrac{\partial w}{\partial z}) \end{aligned} \tag{2.22}$$

$$\begin{aligned} \tau_{xy} = \tau_{yx} &= \mu(\tfrac{\partial v}{\partial x} + \tfrac{\partial u}{\partial y}) \\ \tau_{yz} = \tau_{zy} &= \mu(\tfrac{\partial w}{\partial y} + \tfrac{\partial v}{\partial z}) \\ \tau_{zx} = \tau_{xz} &= \mu(\tfrac{\partial u}{\partial z} + \tfrac{\partial w}{\partial x}) \end{aligned} \tag{2.23}$$

With p as the local thermodynamic pressure. Pleas note that for incompressible flows $\mathrm{div}\ \vec{v} = 0$.

Because the momentum equation (2.19) is a vector equation, the corresponding diffusive term is $\frac{\partial}{\partial x_j}(\mu\frac{\partial u_i}{\partial x_j})$ or $\int_S \mu \ \mathrm{grad}\, u_i \ \cdot \vec{n} \, dS$ (see FP 7.1.1).

One can extract the pressure p from equation (2.22)

2.2 Diffusion term and non-orthogonal meshes

The discretisation of the diffusion term is:

$$\begin{aligned}\int_{V_P} \nabla \cdot (\rho\Gamma_\phi \nabla\phi)\, dV &= \sum_{f=1}^{\text{Nfaces}} \vec{S} \cdot (\rho\Gamma_\phi \nabla\phi)_f \\ &= \sum_{f=1}^{\text{Nfaces}} (\rho\Gamma_\phi)_f \vec{S} \cdot (\nabla\phi)_f . \end{aligned} \tag{2.24}$$

If the mesh is orthogonal then the diffusion term is simply

$$\vec{S} \cdot (\nabla\phi)_f = |\vec{S}| \frac{\phi_N - \phi_P}{|\vec{d}|}. \tag{2.25}$$

Another method would be calculating the cell centered gradients

$$(\nabla\phi)_P = \frac{1}{V_P} \sum_{f=1}^{\text{Nfaces}} \vec{S}\phi_f \tag{2.26}$$

and interpolating to the cell face using (2.11) again:

$$(\nabla\phi)_f = \lambda_f (\nabla\phi)_N + (1 - \lambda_f)(\nabla\phi_)P. \tag{2.27}$$

However this approach cannot be used for non-orthogonal meshes [24].

The product $\vec{S} \cdot (\nabla\phi)_f$ is split into two parts

$$\vec{S} \cdot (\nabla\phi)_f = \underbrace{\vec{s} \cdot (\nabla\phi)_f)}_{\text{orthogonalpart}} + \underbrace{\vec{t} \cdot (\nabla\phi)_f)}_{\text{non-orthogonal corr.}} . \tag{2.28}$$

The two vectors $\vec{s}$ and $\vec{k}$ should give the surface again

$$\vec{S} = \vec{s} + \vec{t}. \tag{2.29}$$

Where $\vec{s}$ simply is parallel to $\vec{d}$ $(= \vec{x}_P - \vec{x}_N)$ and equation (2.25) can still be used.

In [24] the over-relaxed approach is chosen. The contribution of ϕ_P and ϕ_N is increased when the non-orthogonality is increased.

2.3 Calculation of wall shear stresses

The viscous stresses near a wall are ([12] eq. (8.75)):

$$\tau_{nn} = 2\mu \left(\frac{\partial v_n}{\partial n} \right)_{\text{wall}} = 0 \,, \tag{2.30}$$

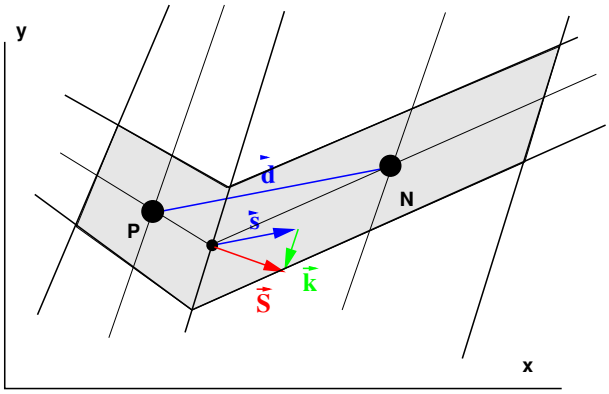


Figure 2.3: Over relaxed non-orthogonal treatment

and

$$\tau_{nt} = \mu \left(\frac{\partial v_t}{\partial n} \right)_{\text{wall}} . \tag{2.31}$$

In which the coordinate t is in the direction of the velocity vector near the wall. The third coordinate s is orthogonal to t and the normal n.

The surface integral of τ_{nt} results in a force:

$$\vec{f}_{\text{wall}} = \int_S \vec{t}\, \tau_{nt}\, dS \;\approx\; (\vec{t}\, \tau_{nt}\, S)_S \tag{2.32}$$

in which $\vec{f}_{\text{wall}}$ points in the direction of $\vec{t}$ (with unit length) and magnitude τ_{nt} multiplied by the surface area S.

2.3.1 Calculation of turbulent wall shear stresses

In subroutine FluxUVW the effective wall shear stress is used. Yet in the law of the wall the shear stress is calculated differently. Start with the laminar formulation:

$$\tau_w = \mu_{eff} \left(\frac{\partial U}{\partial n} \right)_{\text{wall}} \tag{2.33}$$

or

$$\tau_w = \mu_{eff} \frac{U_t}{\Delta n} \tag{2.34}$$

Rewriting

$$\begin{aligned}
\mu_{eff} &= \frac{\tau_w}{U_t/\Delta n} \\
&= \frac{\rho\, u_\tau^2}{U_t/\Delta n} \\
&= \frac{\rho\, u_\tau^2\, \Delta n}{U_t} \\
&= \frac{\rho\, u_\tau\, \Delta n}{\mu}\ \frac{\mu\, u_\tau}{U_t} \\
&= \frac{\rho\, u_\tau\, \Delta n}{\mu}\ \frac{\mu}{U_t/u_\tau} \\
&= \frac{y^+\, \mu}{u^+}
\end{aligned} \tag{2.35}$$

Finally the minimum viscosity possible is the laminar viscosity. Thus a simple test on $y^+/u^+ \geq 1$ is needed.

2.3.2 Rough walls

The law of the wall for a rough wall reads:

$$u^+(z) = \frac{1}{\kappa} \ln\left(\frac{z}{k_s}\right) + B_{\text{rough}}\ , \tag{2.36}$$

with k_s as the sand roughness height, and B_{rough} an empirical constant equal to 8.5 (Schlichting).

Another description (used for atmospheric boundary layers) is:

$$u^+(z) = \frac{1}{\kappa} \ln\left(\frac{z - z_d}{z_0}\right), \tag{2.37}$$

with z_d as the displacement thickness and z_0 an aerodynamic roughness parameter.

In case $z_d = 0$ the relationship between z_0 and k_s is:

$$z_0 = k_s / e^{\kappa B_{\text{rough}}}\ . \tag{2.38}$$

With $B_{\text{rough}} = 8.5$ and $\kappa = 0.4$ follows $z_0 = 1/30\ k_s$, and thus 20 z_0 equals to $2/3\ k_s$.

2.4 Residuals

Consider the equation:

$$A\, \vec{x} = B\ .$$

After some iterations the approximate solution is $\vec{x}^{\,n}$ which is inexact and leaves a non-zero residual:

$$A\,\vec{x}^{\,n} = B - \vec{\rho}^{\,n}\ .$$

or

$$\vec{\rho}^{\,n} = B - A\,\vec{x}^{\,n}\ . \tag{2.39}$$

This is an absolute residue and doe not give a clue of the degree of convergence.

In the subroutine 'Set_Normalisation_Factors' the residuals are normalised using the mass flux entering the domain:

$$\Sigma\dot{m}_{\text{in}} = \sum_{\text{inlets}} \rho_i\,(\vec{u_i}\cdot A_i)\ \ (\equiv M_P) \tag{2.40}$$

For the momentum equations the residual is normalised with the total momentum $m\vec{v}$ entering the domain:

$$M_v = \sum_{\text{inlets}} \dot{m}_i\,\sqrt{\vec{u_i}\cdot\vec{u_i}} \tag{2.41}$$

The turbulence energy:

$$M_k = \sum_{\text{inlets}} \dot{m}_i\,(\vec{u_i}\cdot\vec{u_i}) \tag{2.42}$$

The turbulent dissipation:

$$M_\varepsilon = \frac{M_k}{L/\bar{V}_{\text{inlet}}} \tag{2.43}$$

Note that the latter contains an arbitrary length scale L (and is thus not a very reliable measure).

All the above mentioned normalisation factors depend on the mass flux flowing into the domain. In case there is no inflow then ....

3 Convective term and differencing schemes

The discretisation of the convection term is:

$$\begin{aligned}\int_{V_P} \nabla \cdot (\rho \vec{v}_r \phi)\, dV &= \sum_{f=1}^{\text{Nfaces}} \vec{S} \cdot (\rho \vec{v}_r \phi)_f \\ &= \sum_{f=1}^{\text{Nfaces}} \vec{S} \cdot (\rho \vec{v}_r)_f \phi_f \\ &= \sum_{f=1}^{\text{Nfaces}} F\, \phi_f \end{aligned} \tag{3.1}$$

were one recognises F as the mass flux through the face. Assume that the mass flux is known and the process reduces to interpolating ϕ on the face.

One possibility was already introduced as equation (2.11) and it is called the 'Central Differencing Scheme':

$$\phi_f = \lambda_f \phi_N + (1 - \lambda_f)\phi_P \tag{3.2}$$

with λ defined as:

$$\lambda_f = \frac{x_f - x_P}{x_N - x_P}. \tag{3.3}$$

For non-smooth grids a correction can be applied. This scheme is called 'CD' in dolfyn.

Another central differencing scheme can be defined using the central nodes, the gradients and the distances towards the face center. The last step is averaging the two found values:

$$\phi_f = \frac{1}{2}(\phi_P + \nabla\phi_P(\vec{x}_f - \vec{x}_P) + \phi_N + \nabla\phi_N(\vec{x}_f - \vec{x}_N)). \tag{3.4}$$

This scheme is available as 'CD2'.

Finally a very crude, but simple, method is avaible (just for testing purposes) and that is simply averaging the values of ϕ_P and ϕ_N:

$$\phi_f = \frac{1}{2}(\phi_P + \phi_N). \tag{3.5}$$

Of course this scheme is second order for equidistant grids. This scheme is named 'CD3'.

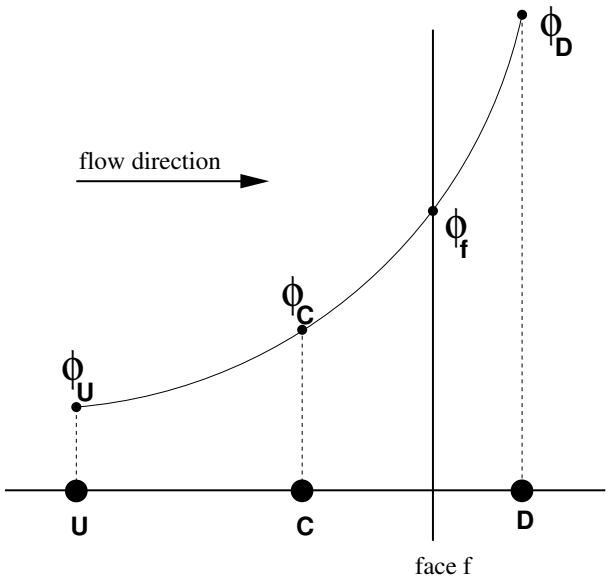


Figure 3.1: Definition of upwind, central and downwind nodes

All the central differencing schemes can cause unphysical, i.e. unbounded, results.

The classic scheme which is guaranteed to be bounded is the 'Standard Upwind Differencing Scheme':

$$\phi_f = \begin{cases} \phi_P & \text{if} \quad (\vec{v}\cdot\vec{n})_f > 0 \\ \phi_N & \text{if} \quad (\vec{v}\cdot\vec{n})_f < 0 \end{cases} . \tag{3.6}$$

See the classic texts and critics on the Upwind Scheme or "pigpen method"[1] of Roache in 1972 [34], or Patankar in 1980 [29].

Next a combination of UDS and CDS is possible by blending the two schemes, or blending any second order scheme with UDS:

$$\phi_f = (1-\gamma)\phi_f^{\text{low order}} + \gamma\phi_f^{\text{high order}}. \tag{3.7}$$

were γ is a fixed constant for the domain.

Although in [12] it is claimed that a blend between UDS and CDS should be suffient, sometimes other bounded high resolution schemes are desired. Following [24][25] and [10] the Normalised Variable Approach (NVA) is introduced. Under certain circumstances boundness can be achieved by introducing a small amount of first order numerical 'diffusion'. First start with the Convection Boundness Criterion for unstructured meshes.

The nomalised variable is defined as (see Figure 3.1):

$$\tilde{\phi} = \frac{\phi - \phi_U}{\phi_D - \phi_U}. \tag{3.8}$$

[1] "D.B. Spalding is purported to have referred to upwind differencing as the "pigpen method," the idea being that if ζ is taken as a concentration fraction, then we should smell the pigpen when we're on the downwind side, and not on the upwind side (except for diffusion effects)." [34]

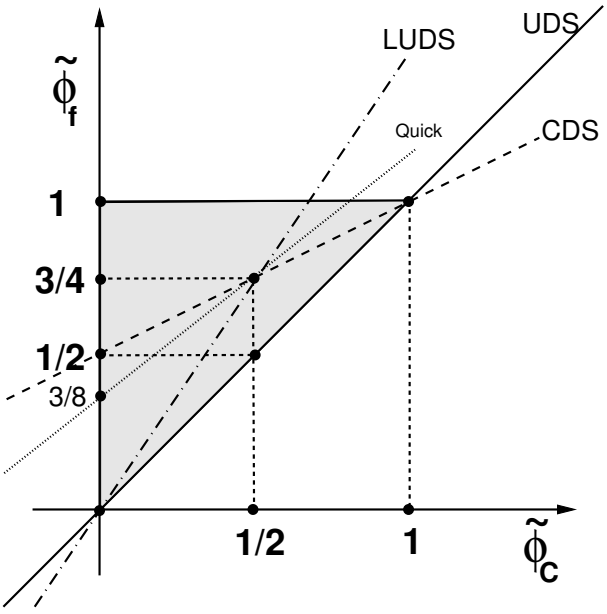


Figure 3.2: Convection Boundness Criterion with UDS, CDS and LUDS

The value at face f is then a function of ϕ at U (upwind node), C (the central node), and D (the downwind node): $\tilde{\phi}_f = f(\tilde{\phi}_C)$ with

$$\tilde{\phi}_C = \frac{\phi_C - \phi_U}{\phi_D - \phi_U} \tag{3.9}$$

and

$$\tilde{\phi}_f = \frac{\phi_f - \phi_U}{\phi_D - \phi_U}. \tag{3.10}$$

Now it is required that ϕ_C and ϕ_f are locally bounded between ϕ_U and ϕ_D, or

$$\phi_U \leq \phi_C \leq \phi_D \tag{3.11}$$

or

$$\phi_U \geq \phi_C \geq \phi_D. \tag{3.12}$$

The Convection Boundness Criterion is then simply:

$$0 \leq \tilde{\phi}_C \leq 1. \tag{3.13}$$

In the Normalised Variable Diagram (NVD) the boundness criteria can be presented, see Figure 3.2, with $\tilde{\phi}_f$ as a function of $\tilde{\phi}_C$. Within the grey region *and on the line* $\tilde{\phi}_f = \tilde{\phi}_C$ the scheme is bounded.

In the CBC the upwind value of ϕ_C is needed for the definition of $\tilde{\phi}_C$. Following [24] again one can argue that ϕ_C is bounded by ϕ_U and ϕ_D and therefore also by the interpolated values at ϕ_f^- and ϕ_f^+ (see Figure 3.3). The new definition of $\tilde{\phi}_C$ is

$$\tilde{\phi}_C = \frac{\phi_C - \phi_f^-}{\phi_f^+ - \phi_f^-} = 1 - \frac{\phi_f^+ - \phi_C}{\phi_f^+ - \phi_f^-}. \tag{3.14}$$

Recall Equation (2.11):

$$\phi_f = \lambda_f \phi_N + (1 - \lambda_f)\phi_P \tag{3.15}$$

with λ defined as:

$$\lambda_f = \frac{x_f - x_P}{x_N - x_P}. \tag{3.16}$$

Thus $\phi_f^+ - \phi_C$ is (with $\phi_N = \phi_D$ and $\phi_P = \phi_C$)

$$\begin{aligned} \phi_f^+ - \phi_C &= \lambda_f \phi_D + (1 - \lambda_f)\phi_C - \phi_C \\ &= \lambda_f \phi_D - \lambda_f \phi_C \\ &= \lambda_f (\phi_D - \phi_C) \\ &= \lambda_f \frac{\phi_D - \phi_C}{x_N - x_C} (x_N - x_C) \\ &= \lambda_f \ (\nabla\phi)_f \cdot \hat{\mathbf{d}} \ (x_N - x_C). \end{aligned} \tag{3.17}$$

With the unit vector $\hat{\mathbf{d}}$ parallel with $\overline{CD}$

$$\hat{\mathbf{d}} = \frac{\vec{\mathbf{d}}}{|\mathbf{d}|}. \tag{3.18}$$

For the two faces ϕ_f^+ and the *virtual upstream face* ϕ_f^- the process is analogous:

$$\begin{aligned} \phi_f^+ - \phi_f^- &= \frac{\phi_f^+ - \phi_f^-}{x_f^+ - x_f^-} (x_f^+ - x_f^-) \\ &= (\nabla\phi)_C \cdot \hat{\mathbf{d}} \ (x_f^+ - x_f^-). \end{aligned} \tag{3.19}$$

Now Equation 3.14 turns into

$$\begin{aligned} \tilde{\phi}_C &= 1 - \frac{\phi_f^+ - \phi_C}{\phi_f^+ - \phi_f^-} \\ &= 1 - \frac{\lambda_f \ (\nabla\phi)_f \cdot \hat{\mathbf{d}} \ (x_D - x_C)}{(\nabla\phi)_C \cdot \hat{\mathbf{d}} \ (x_f^+ - x_f^-)} \end{aligned} \tag{3.20}$$

Using the definition of λ which is defined as the distance from point C to ϕ_f^+ divided by $\overline{CD}$) and the fact that C is located in the middle of the cell gives

$$\frac{x_f^+ - x_f^-}{x_D - x_C} = 2 \frac{x_f^+ - x_C}{x_D - x_C} = 2\lambda_f \tag{3.21}$$

and Eqn. (3.20) transforms into

$$\begin{aligned} \tilde{\phi}_C &= 1 - \frac{(\nabla\phi)_f \cdot \hat{\mathbf{d}}}{2 \ (\nabla\phi)_C \cdot \hat{\mathbf{d}}} \\ &= 1 - \frac{(\nabla\phi)_f \cdot \mathbf{d}}{2 \ (\nabla\phi)_C \cdot \mathbf{d}} \\ &= 1 - \frac{\phi_D - \phi_C}{2 \ (\nabla\phi)_C \cdot \mathbf{d}} \end{aligned} \tag{3.22}$$

because the product $(\nabla\phi)_f \cdot \mathbf{d}$ is equal to $\phi_D - \phi_C$.

Using this framework all kinds of differencing schemes can be defined based on the value of $\tilde{\phi}_C$.

3.1 Standard upwind differencing, UD

In this case simply use the upwind value.

$$\phi_f = \phi_C. \tag{3.23}$$

3.2 Second order central differencing, CD

3.2.1 Linear interpolation, CD1

The standard method proposed by [12] is the linearly weighted central differencing scheme. Including the non-orthogonal correction it reads

$$\phi_f = \lambda_f \phi_D + (1 - \lambda_f)\phi_C + (\nabla\phi)_f(\vec{x}_{f'} - \vec{x}_f). \tag{3.24}$$

The orginal formulation does not use the Convection Boundness Criterion (CBC).

Central differencing is important to LES simulations.

3.2.2 Gradient based averaging, CD2

Another CDS method is based on averaging the results including both the up- and downstream gradients as in

$$\phi_f = \frac{1}{2}\left(\phi_C + \nabla\phi_C(\vec{x}_f - \vec{x}_C) + \phi_D + \nabla\phi_D(\vec{x}_f - \vec{x}_D)\right). \tag{3.25}$$

The attractivness of this formulation is that it automatically deals with non-orthogonal, or skewed, cells.

3.2.3 Simple averaging, CD3

The most simple method, discarding all the geometrical information.

$$\phi_f = \frac{1}{2}(\phi_C + \phi_D). \tag{3.26}$$

3.3 Blended differencing

3.3.1 Constant blending

Using a fixed blending, by an amount of γ, between UD and CDS is the proposed method in [12]:

$$\phi_f = \gamma\phi_{\mathrm{CD}} + (1 - \gamma)\phi_{\mathrm{UD}}. \tag{3.27}$$

3.3.2 Variable blending, hybrid scheme

Instead of using a fixed blending factor in the domain one could differentiate on the basis of the local Peclet number Pe. The Peclet number is defined as

$$Pe = \frac{(\rho\, u)_f\, \Delta x_f}{\Gamma_f}. \tag{3.28}$$

If the viscosity is used for the diffusion coefficient Γ it is immedeately obvious that the Peclet number remsembles the Reynolds number Re. Hence the term 'cell Reynolds number' is often used as well.

In [12] and [29] it is shown that the oscillations depend on the value of Pe. When $Pe \leq 2$ at every grid node, no oscillations occur. 'This is a suffient, but not necessary condition for boundness of CDS solution' [12]. Spaldings hybrid scheme [38] was designed to switch from CDS to UD at any node at which $Pe \geq 2$. 'This is too restrictive and reduces the accuracy' [12] as it is only needed in regions where the gradient is high. Another way of looking at it is to use the viscosity and regard it as the cell Reynolds number; even for very mild air flow the Peclet number would jump up and UD would be used everywhere. For a further discusion see [12].

The hybrid scheme is therefore not implemented.

3.3.3 Variable blending, power law scheme

Patankar introduced a scheme based on the exact one-dimensional solution which should provide better results than the hybrid scheme [29]. It switches to UD when $Pe \geq 10$.

The original formulation is of the form [29]

$$\frac{\phi(x) - \phi_0}{\phi_L - \phi_0} = \frac{\exp(Pe\frac{x}{L}) - 1}{\exp(Pe) - 1} \tag{3.29}$$

hence the name 'power law'.

In [42] an equivalent expression is given:

$$\phi_f = \phi_C - \beta_f(\phi_D - \phi_C) \tag{3.30}$$

for $0 < Pe < 10$ with

$$\beta_f = \frac{(1 - 0.1Pe_f)^5}{Pe_f}. \tag{3.31}$$

Especially the original formulation should not be used as the numerous calls to the exponentional function would be too costly. The power law scheme, although it performs better than the hybrid scheme, inherits the same defficiencies. The power law scheme is not implemented.

3.4 Second order linear upwind differencing, LUD

The original formulation includes a second upwind node in order to find a gradient and extrapolate to the face. As gradients are available the same idea can be accomplished using:

$$\phi_f = \phi_C + \nabla\phi_C \, (\vec{x}_f - \vec{x}_C). \tag{3.32}$$

Two forms are implemented: LUD using the CBC framework (and reverting to UD outside the range) and the unbounded version LUX.

3.5 Third order linear upwind differencing

Historically the third order linear upwind differencing 'Quadratic Upwind Interpolation for Convective Kinematics', or QUICK is important. In principle this scheme involves an additional upstream node (or $\phi_U U$). The search for such a 'virtual' node is not useful. Moreover the results in the past of QUICK were rarely superior.

In the NVD diagram, see Figure 3.2, the QUICK scheme is found to be equal to [2]

$$\phi_f = \frac{3}{8} + \frac{3}{4}\phi_C. \tag{3.33}$$

The QUICK scheme is not implemented.

3.6 Second order switching schemes

3.6.1 MinMod differencing, MinMod

Following Alves et.al. [1]

3.7 Other schemes

3.7.1 Third order monotone upstream, MUSCL

The third order 'Monotone Upstream-centered Scheme for Conservation Laws', or MUSCL, is a blending of a CDS scheme and the LUD scheme. In short

$$\phi_f = \gamma\phi_{\mathrm{CDS}} + (1-\gamma)\phi_{\mathrm{LUD}}. \tag{3.34}$$

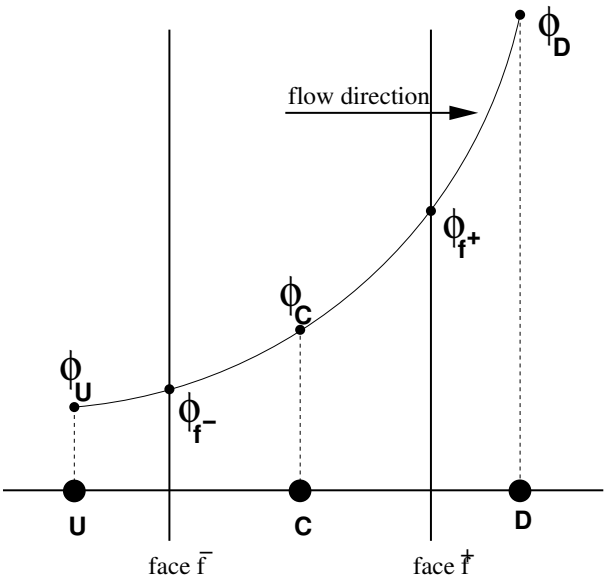


Figure 3.3:

It has not (yet) been implemented. It can be easily done, either with or without the CBC framework.

3.7.2 Gamma differencing, GAM

LUDS

MinMod

GAMMA

The schemes are not complete yet. On *very skewed*, really bad or dirty, meshes instabilities still occur. This point needs further attention as a remedy has not been found yet.

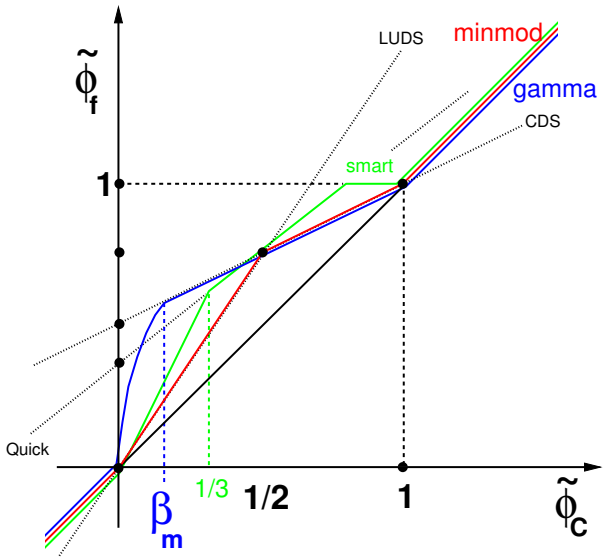


Figure 3.4: Smart, MinMod and Gamma-schemes

4 Calculation of gradients

Currently only two methods have been used: Gauss and Least Squares. Both are called from subroutine *GrandientPhi*.

Both depend on the number of neighbours; the more the merrier (and the accuracy improves). And here tetraeders, especially near boundaries, are suffering from the reduced number of neighbours.

4.1 Gauss

Consider Gauss' divergence theorem applied to a gradient:

$$\begin{aligned} \int_V \text{grad}\ \psi\ dV &= \int_S \psi\ d\vec{s} \\ (\text{grad}\ \psi)_{P_0} &\approx \frac{1}{V_{P_0}} \sum_{j=1}^{n_f} \psi_j \vec{s}_j \end{aligned} \tag{4.1}$$

with ψ_j the value at the centre of face j.

This is done in subroutine *GradientPhiGauss*.

4.2 Least squares

Another possibility is based on the following thought: one knows the values at the cell centre surrounding point P_0. Thus for one pair

$$\vec{d}_j \cdot (\text{grad}\ \psi)_{P_0} = \psi_{P_j} - \psi_{P_0}$$

or the set

$$\vec{d}_j \cdot (\text{grad}\ \psi)_{P_0} = \psi_{P_j} - \psi_{P_0} \quad (j = 1, \ldots, n) \tag{4.2}$$

with $\vec{d_j} = \vec{x}_{P_j} - \vec{x}_{P_0}$ the vector from P_0 to P_j. This results in general to more sets or pairs than there are unknowns. For example

$$\begin{aligned}
d_{j+1_1}\frac{\delta\psi}{\delta x_1} + d_{j+1_2}\frac{\delta\psi}{\delta x_2} + d_{j+1_3}\frac{\delta\psi}{\delta x_3} &= \psi_{P_{j+1}} - \psi_{P_0} \\
d_{j+2_1}\frac{\delta\psi}{\delta x_1} + d_{j+2_2}\frac{\delta\psi}{\delta x_2} + d_{j+2_3}\frac{\delta\psi}{\delta x_3} &= \psi_{P_{j+2}} - \psi_{P_0} \\
d_{j+3_1}\frac{\delta\psi}{\delta x_1} + d_{j+3_2}\frac{\delta\psi}{\delta x_2} + d_{j+3_3}\frac{\delta\psi}{\delta x_3} &= \psi_{P_{j+3}} - \psi_{P_0} \\
d_{j+4_1}\frac{\delta\psi}{\delta x_1} + d_{j+4_2}\frac{\delta\psi}{\delta x_2} + d_{j+4_3}\frac{\delta\psi}{\delta x_3} &= \psi_{P_{j+4}} - \psi_{P_0} \\
&\vdots \\
d_{j+n_1}\frac{\delta\psi}{\delta x_1} + d_{j+n_2}\frac{\delta\psi}{\delta x_2} + d_{j+n_3}\frac{\delta\psi}{\delta x_3} &= \psi_{P_{j+n}} - \psi_{P_0}
\end{aligned}$$

With the unknowns $\frac{\delta\psi}{\delta x_1}$, $\frac{\delta\psi}{\delta x_2}$, and $\frac{\delta\psi}{\delta x_3}$. Now multiply equation (4.2) with the transpose of $\vec{d_j}$:

$$\vec{d_j}^T\vec{d_j} \cdot \begin{pmatrix} \frac{\delta\psi}{\delta x_1} \\ \frac{\delta\psi}{\delta x_2} \\ \frac{\delta\psi}{\delta x_3} \end{pmatrix}_{P_0} = \vec{d_j}^T \Delta\psi_{P_{j,0}} \quad (j = 1, \ldots, n) \tag{4.3}$$

And the sum of (4.3) is:

$$\sum_{j=1}^{n_f} \vec{d_j}^T\vec{d_j} \cdot \begin{pmatrix} \frac{\delta\psi}{\delta x_1} \\ \frac{\delta\psi}{\delta x_2} \\ \frac{\delta\psi}{\delta x_3} \end{pmatrix}_{P_0} = \sum_{j=1}^{n_f} \vec{d_j}^T \Delta\psi_{P_{j,0}} \quad (j = 1, \ldots, n) \tag{4.4}$$

Or

$$\sum_{j=1}^{n_f} \begin{pmatrix} {\delta x_1}^2 & \delta x_2\delta x_1 & \delta x_3\delta x_1 \\ \delta x_1\delta x_2 & {\delta x_2}^2 & \delta x_3\delta x_2 \\ \delta x_1\delta x_3 & \delta x_2\delta x_3 & {\delta x_3}^2 \end{pmatrix}_j \cdot \begin{pmatrix} \frac{\delta\psi}{\delta x_1} \\ \frac{\delta\psi}{\delta x_2} \\ \frac{\delta\psi}{\delta x_3} \end{pmatrix}_{P_0} = \sum_{j=1}^{n_f} \begin{pmatrix} \delta x_1\Delta\psi_{P_{j,0}} \\ \delta x_2\Delta\psi_{P_{j,0}} \\ \delta x_3\Delta\psi_{P_{j,0}} \end{pmatrix}_j$$

This looks like the classic:

$$A\,\vec{x} = B$$

With

$$A = \sum_{j=1}^{n_f} \vec{d_j}^T\vec{d_j} \tag{4.5}$$

and

$$B = \sum_{j=1}^{n_f} \vec{d_j}^T(\psi_{P_j} - \psi_{P_0}) \tag{4.6}$$

and the solution is $\vec{x} = (\text{grad}\ \psi)_{P_0}$.

This is done in subroutine *GradientPhiLeastSquares*.

Note that A is a symmetric 3 x 3 matrix with only geometrical coefficients (the distances from P_0 to P_j). So inverting A gives:

$$(\text{grad } \psi)_{P_0} = A^{-1} \sum_{j=1}^{n_f} \vec{d}_j^T (\psi_{P_j} - \psi_{P_0}) \tag{4.7}$$

A^{-1} can be stored once and the gradient is simply calculated every time from equation (4.7). Of course storing will need some extra memory (6 elements per cell).

One should be aware that Gauss Elimination without pivoting is numerically unstable.

On the other hand, the Gauss Elimination method becomes stable when using pivoting. This means that one can interchange rows (partial pivoting) or rows and columns (full pivoting) to put a suitable matrix element to the position of the current pivot. Since it is desirable to keep the already constructed part of the identify matrix, only rows below and columns right to the current pivot are considered. Gauss Elimination with full pivoting is numerically stable. From the application point of view, Gauss Elimination with partial pivoting is numerically stable too, although there are artificial examples where it fails. Additionally, the advantage of partial pivoting (compared to full pivoting) is that it does not change the order of solution components.

5 Heat transfer

For non-isothermal flows the energy equation reads:

$$\underbrace{\frac{\partial}{\partial t}(\rho\, c_p T)}_{\text{unsteady term}} + \underbrace{\text{div}(\,\rho\,\vec{v}_r\, c_p T\,)}_{\text{convective term}} = \underbrace{\text{div}(\,\lambda\,\text{grad}\,T\,)}_{\text{diffusive term}} + \underbrace{q_T}_{\text{source terms}} \tag{5.1}$$

The primitive variable is the temperature T thus in order to adhere to the standard scalar transport equation the heat capacity is rearranged to:

$$\frac{\partial}{\partial t}(\rho\, T) + \text{div}(\,\rho\,\vec{v}_r\, T\,) = \text{div}(\,\frac{\lambda}{c_p}\,\text{grad}\,T\,) + \frac{q_T}{c_p} \tag{5.2}$$

Where λ/c_p appears as the thermal diffusivity Γ_T.

Near solid walls either the wall temperature T_w or the heat flux q_w through the wall can be specified. In both cases the heat transfer coefficient is needed.

The heat transfer coefficient h is a simple way to lump the heat transfer in one empirical formula. Starting from

$$\dot{Q} = h\; A\; (T_w - T)$$

and rearranging

$$h = \frac{\dot{Q}/A}{T_w - T} = \frac{q_w}{T_w - T} \tag{5.3}$$

Thus once the heat flux q_w and the temperatures T_w and T are known the heat transfer coefficient is defined by equation (5.3). Likewise if instead of the temperature in the cell adjacent to the wall T some other reference or bulk temperature T_{bulk} is used, another value of h is found. This latter value should correspond to the values found in empirical handbooks of heat transfer.

Recall that λ/c_p is used as the thermal diffusivity in the fluid domain. Introducing the (laminar) Prandtl number:

$$Pr = \frac{c_p\,\mu}{\lambda} \tag{5.4}$$

hence

$$\frac{\lambda}{c_p} = \frac{\mu}{Pr} \tag{5.5}$$

which links and scales the laminar viscosity directly to the thermal diffusivity.

For an given wall temperature T_w and heat transfer coefficient h the heat source $\dot{Q} = q_w\; A = h\; A\; (T_w - T)$ near the boundary cell is known and splits in two

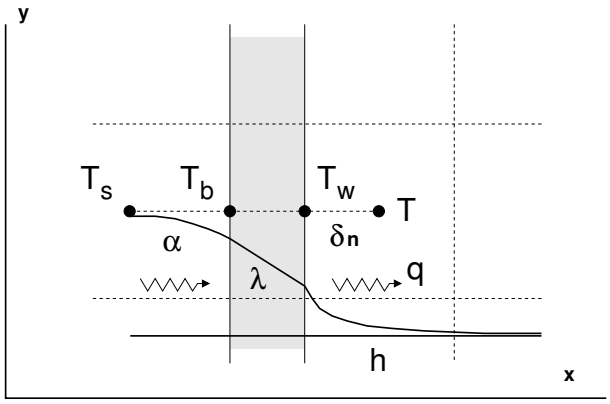


Figure 5.1: Heat transfer near a wall

parts: $h\ A\ T_w$ and $h\ A\ T$, the former enters the source term S_T for the boundary cell and the latter enters the diagonal term A_p as $h\ A$.

When the heat flux q_w is used as a boundary condition then the wall temperature is first calculated. This value is then used in the source term.

The temperature boundary condition T_w does not need to be specified at the solid fluid interface. Assuming one dimensional heat conduction through a wall of s thick and a heat conduction of λ_s gives (see Figure 5.1):

$$q = \frac{\lambda_s}{s}\ (T_b - T_w) \tag{5.6}$$

In the surroundings with T_s and an assumed heat transfer coefficient of α_s results in:

$$q = \alpha_s\ (T_s - T_b) \tag{5.7}$$

The heat flux through all three components must be equal thus

$$q = \alpha_s\ (T_s - T_b) = \frac{\lambda_s}{s}\ (T_b - T_w) = h\ (T_w - T). \tag{5.8}$$

The overall temperature difference is then:

$$\begin{aligned} T_s - T &= (T_s - T_b) + (T_b - T_w) + (T_w - T) \\ &= \frac{q}{\alpha_s} + \frac{q}{\lambda/s} + \frac{q}{h} \\ &= q\ (\frac{1}{\alpha_s} + \frac{1}{\lambda/s} + \frac{1}{h}) \end{aligned} \tag{5.9}$$

and can be regarded as a chain of resistances as in $q = \Delta T / \sum R_i$ or

$$q = \frac{\Delta T}{\frac{1}{h} + \sum R_i}. \tag{5.10}$$

$$R = \frac{s}{\lambda}$$

5.1 Laminar heat transfer

In the case of laminar flow the heat transfer is dictated by heat conduction from the centre of the wall cell to the wall over a distance of δn. The heat flux simply reads:

$$\begin{aligned} q &= \lambda \frac{(T_w - T)}{\delta n} \\ &= \frac{\mu\, c_p}{Pr} \frac{(T_w - T)}{\delta n} \\ &= \frac{1}{\frac{Pr\ \delta n}{\mu\ c_p}} (T_w - T) \\ &= \frac{1}{\frac{1}{h} + \sum R_i} (T_w - T). \end{aligned} \tag{5.11}$$

With the local heat transfer coefficient h

$$h = \frac{\mu\, c_p}{Pr\ \delta n}. \tag{5.12}$$

Note that λ / c_p is the thermal diffusivity Γ_T in the transport equation. Also the heat source (or sink) for the boundary cell is needed. In the code itself

$$\dot{Q} = H\ (T_w - T) \tag{5.13}$$

is used, with

$$H = \frac{1}{\frac{1}{\mu/(Pr\delta n)} + c_p\ \sum R_i} A. \tag{5.14}$$

However for output purposes H has to be transferred into h again

$$h = H\ \frac{c_p}{A} \tag{5.15}$$

with a heat flux in W/m^2

$$q_w = h\ (T_w - T) \tag{5.16}$$

and a total heat balance of

$$Q = \sum q_{w_i}\ A_i. \tag{5.17}$$

5.2 Turbulent heat transfer

The turbulent heat transfer is basically modelled analog to the viscous boundary layer. Where the law of the wall profile $u^+ = f(y^+)$ is scaled by the turbulent Prandtl number Pr_t (or σ_T). In short dimensionless form:

$$T^+ = Pr_t\ u^+ \tag{5.18}$$

However, in general a sublayer resistance factor (commonly denoted by P) by C. Jayatilleke is accounted for as well:

$$P = 9.24 \Big[\left(\frac{Pr}{Pr_t} \right)^{3/4} - 1 \Big] \Big[1 + 0.28 \exp \left(\frac{-0.007\ Pr}{Pr_t} \right) \Big] \tag{5.19}$$

which brings eqn. (5.18) to

$$T^+ = Pr_t \, (u^+ + P) \tag{5.20}$$

The dimensionless temperature T^+ is defined by:

$$T^+ = \frac{\rho \, \mu_t \, (T_w - T)}{q_w} \tag{5.21}$$

Using the definition of the heat transfer coefficient (5.3), and eqns. (5.20) and (5.21) gives

$$h = \frac{q_w}{T_w - T} = \frac{\rho \, C_p \, \mu_t}{Pr_t(u^+ + P)} \tag{5.22}$$

One of the great consequences of eqn. (5.22) is that one does not need to perform a thermal simulation in order to find the local heat transfer coefficients. On the other hand, a local heat transfer might become high, but when the temperature difference $T_w - T$ is negligible, there will hardly be any heat flux and thus no cooling (or warming up).

6 Lagrangian particles

6.1 Basic equations

Multi-phase flows often contain particles or droplets. Let $\vec{u}$ be the velocity of the continuous phase, $\vec{v}_p$ the current droplet velocity and $\vec{x}_p$ the droplet position.

Then the Newtonian equation of the motion of a droplet with mass m_p is

$$m_p \frac{d\vec{v}_p}{dt} = \sum_i \vec{F}_i \; . \tag{6.1}$$

After summing all the forces the equation of motion (6.1) can be solved using

$$\frac{d\vec{x}_p}{dt} = \vec{v}_p \; . \tag{6.2}$$

6.1.1 Drag force

The most important force is the drag force

$$\vec{F}_d = C_d(Re) \, \frac{1}{2} \rho_f \, |\vec{u} - \vec{v}_p| (\vec{u} - \vec{v}_p) \, S_p \tag{6.3}$$

with the drag coefficient C_d as a function of the droplet Reynolds number Re. S_p is the frontal area of the particle.

The droplet Reynolds number is defined by

$$Re_p = \frac{\rho_f \, |\vec{u} - \vec{v}_p| \, D_p}{\mu} \; . \tag{6.4}$$

The drag coefficient is

$$Re_p \begin{cases} 24,000. & , \quad Re \leq 0.001 \\ 24/Re & , \quad 0.001 \leq Re \leq 0.2 \\ 24 \, (1 + 0.15 \, Re_p^{0.687})/Re_p & , \quad 0.2 < Re \leq 1000 \\ 0.44 & , \quad 1000 < Re \leq 200\,000. \\ 0.10 & , \quad Re > 200\,000. \end{cases} \tag{6.5}$$

6.1.2 Virtual mass force

For accelerating or decelerating particles the surrounding carrier fluid needs to be accelerated or decelerated too. Potential flow theory, also know as irrotational flow (without viscosity), shows that an accelerating particle give rise to a force of:

$$\vec{F}_v = -C_v\, \rho_f \Omega_p\, \frac{d(\vec{v}_p - \vec{u})}{dt}\,. \tag{6.6}$$

Using potential flow theory the 'virtual mass coefficient' C_v is found to be equal to 0.5.

For a steady state flow equation (6.6) reduces to

$$\vec{F}_v = -C_v\, \rho_f \Omega_p\, \frac{d\vec{v}_p}{dt}\,. \tag{6.7}$$

The mass of a particle m_p then simply increases by $C_v\, \rho_f \Omega_p$ to $(\rho_p + C_v\, \rho_f)\Omega_p$. Thus for a particle with density ρ_p of 1000. kg/m^3 in air ($\rho_f \approx 1.2$ kg/m^3) the increase is neglible (just 0.06%).

6.1.3 Body forces

Gravity, or any other external force, can also act on a particle. It is incorporated into the 'body force'. When gravity acts on a particle it will also let a particle float or the buoyant force is equal to the weight of the displaced fluid (Archimedes's principle). The net result is

$$\vec{F}_b = m_p\, \vec{g}\; = (\rho_p - \rho_f)\Omega_p\, \vec{g}\,. \tag{6.8}$$

6.1.4 Adding all forces

Starting with equation (6.1) and using $S_p = \frac{3}{2}\Omega_p/D_p$, $\rho'_p = \rho_p + C_v\rho_f$ and $\vec{v}_s = \vec{u} - \vec{v}_p$:

$$\begin{aligned}
m_p \frac{d\vec{v}_p}{dt} &= \sum_i \vec{F}_i \\
\rho_p \Omega_p \frac{d\vec{v}_p}{dt} &= \vec{F}_d + \vec{F}_v + \vec{F}_b \\
(\rho_p + C_v\rho_f)\Omega_p \frac{d\vec{v}_p}{dt} &= C_d\, \frac{1}{2}\rho_f\, \|\vec{v}_s\|\, (\vec{u} - \vec{v}_p)\, \frac{3}{2}\frac{\Omega_p}{D_p} + (\rho_p - \rho_f)\, \Omega_p\, \vec{g} \\
\rho'_p \frac{d\vec{v}_p}{dt} &= C_d\, \frac{1}{2}\rho_f\, \|\vec{v}_s\|\, (\vec{u} - \vec{v}_p)\, \frac{3}{2}\frac{1}{D_p} + (\rho_p - \rho_f)\, \vec{g} \\
\frac{d\vec{v}_p}{dt} &= C_d\, \frac{3}{4}\frac{\rho_f}{\rho'_p\, D_p}\, \|\vec{v}_s\|\, (\vec{u} - \vec{v}_p) + \frac{\rho_p - \rho_f}{\rho'_p}\, \vec{g}
\end{aligned} \tag{6.9}$$

Equation (6.9) can be solved analytically provided some terms are assumed to be constant:

$$\frac{d\vec{v}_p}{dt} = C_1\, (\vec{u} - \vec{v}_p) + C_2\, \vec{g} \tag{6.10}$$

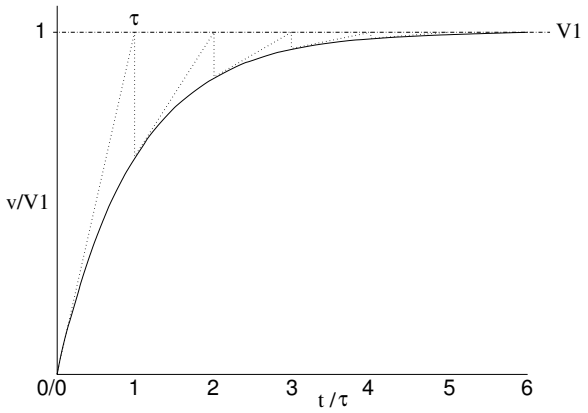


Figure 6.1: Characteristic response of a particle to forces

with

$$C_1 = C_d \, \frac{3}{4} \frac{\rho_f}{\rho'_p \, D_p} \, \|\vec{v}_s\| \tag{6.11}$$

and

$$C_2 = \frac{\rho_p - \rho_f}{\rho'_p} \, . \tag{6.12}$$

Rearanging 6.10 to

$$\frac{d\vec{v}_p}{C_1 \, (\vec{u} - \vec{v}_p) + C_2 \, \vec{g}} = dt$$

and integrating

$$\begin{aligned} \frac{-1}{C_1} \ln(\, C_1 \, (\vec{u} - \vec{v}_p) + C_2 \, \vec{g} \,) &= t + c \\ C_1 \, (\vec{u} - \vec{v}_p) + C_2 \, \vec{g} &= e^{-C_1 t + c} \\ C_1 \, (\vec{u} - \vec{v}_p) + C_2 \, \vec{g} &= C \, e^{-C_1 t} \end{aligned}$$

or

$$\vec{v}_p = (\vec{u} + \frac{C_2}{C_1} \, \vec{g}) - \frac{C}{C_1} \, e^{-C_1 t} \tag{6.13}$$

At $t = 0$ the particle has an initial velocity of $\vec{v}_{p_0}$ thus C is:

$$\vec{v}_{p_0} = C_1 \, (\vec{u} - \vec{v}_p) + C_2 \, \vec{g} - \frac{C}{C_1}$$

or

$$C = (\, C_1 \vec{u} + C_2 \vec{g} \,) - C_1 \vec{v}_{p_0} \, .$$

The particle velocity is then

$$\vec{v}_p = \underbrace{(\, \vec{u} + \frac{C_2}{C_1} \, \vec{g} \,)}_{\vec{v}_1} - \underbrace{(\, \vec{u} + \frac{C_2}{C_1} \, \vec{g} - \vec{v}_{p_0} \,)}_{\vec{v}_2} \, e^{-C_1 t} \tag{6.14}$$

and is illustrated in Figure 6.1. In a quiescent fluid ($\vec{u} = 0$) and after some time a particle will drop subjected to the action of gravitational force $\vec{g}$

$$\vec{v}_{p_\infty} = \frac{C_2}{C_1} \, \vec{g}$$

or

$$\vec{v}_{p_\infty} = \frac{(\rho_p - \rho_f)\, D_p\, \vec{g}}{\frac{3}{4} C_d\, \rho_f\, \|\vec{v}_s\|}\,.$$

For small particles the drag coefficient equals to $24/Re$. Thus with $\|\vec{v}_s\| = \|\vec{u} - \vec{v_p}\| = \|\vec{v_p}\|$

$$\begin{aligned}
\vec{v}_{p_\infty} &= \frac{(\rho_p - \rho_f)}{\frac{24}{Re}\frac{3}{4}\frac{\rho_f}{D_p}\,\|\vec{v}_p\|}\,\vec{g} \\
&= \frac{(\rho_p - \rho_f)}{\frac{24\mu}{\rho_f \|\vec{v}_p\| D_p}\frac{3}{4}\frac{\rho_f}{D_p}\,\|\vec{v}_p\|}\,\vec{g} \\
&= \frac{(\rho_p - \rho_f)}{\frac{18\mu}{D_p}\frac{1}{D_p}}\,\vec{g} \\
&= \frac{(\rho_p - \rho_f)}{18\mu} D_p^2\,\vec{g}\,. \qquad (6.15)
\end{aligned}$$

Which is the well known particle settling or sedimentation velocity for small particles or 'Stokes' law'.

The characteristic time in the response of equation (6.14) can be found using

$$\vec{v}_{p_\infty} = \vec{v}_{p_o} + \left.\frac{d\vec{v_p}}{dt}\right|_{t=0} \Delta t$$

or

$$\vec{v}_{p_\infty} = \vec{v}_{p_o} + (C_1\,(\vec{u} - \vec{v}_p\,) + C_2\,\vec{g}\,)\;\Delta t$$

and consider again a quiescent fluid ($\vec{u} = 0$) and a starting velocity $\vec{v}_{p_0} = 0$.

The characteristic time τ_{momentum} or τ_M is then

$$\begin{aligned}
\frac{C_2}{C_1}\,\vec{g} &= \vec{v}_{p_o} + (C_1\,(\vec{u} - \vec{v}_p\,) + C_2\,\vec{g}\,)\;\Delta t \\
\frac{C_2}{C_1}\,\vec{g} &= C_2\,\vec{g}\,\Delta t \\
\Delta t &= \frac{1}{C_1} \\
\tau_M &= \frac{4\,\rho'_p\,D_p}{3\,C_d\,\rho_f\,\|\vec{v}_s\|} \qquad (6.16)
\end{aligned}$$

which meaning becomes clear in Figure 6.1. Note that the gravitational effects are cancelled out and the characteristic time is only based on the drag force.

For light or small particles in the Stokes' range equation (6.16) reduces to

$$\begin{aligned}
\tau_M &= \frac{4\,\rho'_p\,D_p}{3\,\frac{24}{Re}\,\rho_f\,\|\vec{v}_s\|} \\
&= \frac{4\,\rho'_p\,D_p}{3\,\frac{24\mu}{\rho_f\,\|\vec{v}_s\|\,D_p}\,\rho_f\,\|\vec{v}_s\|} \\
&= \frac{\rho'_p\,D_p^2}{18\,\mu} \qquad (6.17)
\end{aligned}$$

note that the division of $|\vec{u} - \vec{v}_p|/|\vec{u} - \vec{v}_p|$ which goes to $0/0$ when the slip velocity drops to zero has been eliminated.

6.2 Integrating particles in time

The most common method to integrate equation (6.2) is to use a simple explicit method known as 'explicit Euler' or 'forward Euler'. It is so popular because it relies only on the currently known values at time level t^n:

$$\phi^{n+1} = \phi^n + f(\phi^n, t^n)\ \Delta t \tag{6.18}$$

It is only first order accurate but more important it has restrictions in the size of Δt for stability (set by τ_M).

Equation (6.17) also shows what happens when very light and small particles are going to be used. Consider a tiny ($D_p = 1\ 10^{-6}$ m) droplet of water ($\rho_p = 1000.$ kg/m^3) in air ($\mu = 18\ 10^{-6}$ Pa s) when τ_M will drop to $\approx 3\ 10^{-6}$ s. Therefore explicit Euler has it's limitations when any kind of droplet has to be catered for.

An alternative is to use 'implicit Euler' or 'backward Euler' method

$$\phi^{n+1} = \phi^n + f(\phi^{n+1}, t^{n+1})\ \Delta t \tag{6.19}$$

in which there is, in principle, no restriction to the time step size Δt. The disadvantage is that the values of $f(\phi^{n+1}, t^{n+1})$ at the new time t^{n+1} are unknown and extra work is necessary.

When equation (6.9) is casted in the form of (6.10)

$$\frac{d\vec{v}_p}{dt} = C_1\ (\vec{u} - \vec{v}_p) + C_2\ \vec{g}$$

with only $\vec{v}_p$ and t as variables then it can be solved analytically. The solution is:

$$\vec{v}_p = \vec{v}_1 + \vec{v}_2\ e^{-C_1 t} \tag{6.20}$$

with

$$\vec{v}_1 = \vec{u} + \frac{C_2}{C_1}\ \vec{g}$$

and

$$\vec{v}_2 = \vec{u} + \frac{C_2}{C_1}\ \vec{g} - \vec{v}_{p_0} = v\vec{1} - \vec{v}_{p_0}$$

and the constants C_1 and C_2 defined in 6.11 and 6.12 and the characteristic time $\tau_M = 1/C_1$.

The solution of $\vec{v}_p$ in equation (6.20) approaches $\vec{v}_1$ rapidly (see Figure 6.1). How close $\vec{v}_p$ is to the final solution $\vec{v}_1$ depends on it's initial velocity $\vec{v}_{p_0}$, or the ratio of $\vec{v}_{p_0}/\vec{v}_1$. The closer the ratio is to 1 the larger the time step can be allowed to be.

The velocity 6.20 can be integrated once again to obtain the new position:

$$\vec{x}_p = \vec{v}_1\ t + \frac{\vec{v}_2}{C_1}\ (1 - e^{-C_1 t}) \tag{6.21}$$

xxxxx

The basic scheme is:

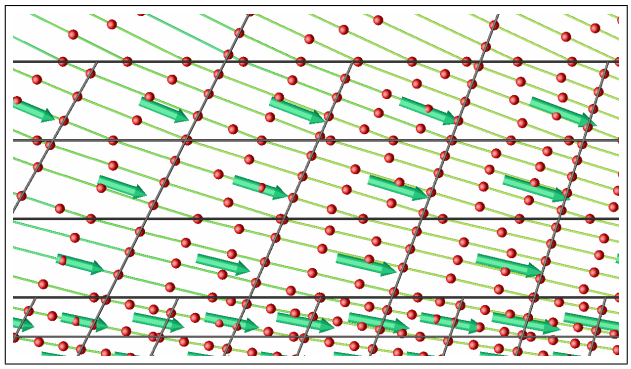


Figure 6.2: Example of particles passing through cells

1. choose a suitable time step Δt
2. use the current particle velocity V_p, or fluid velocity U (whichever is higher), to get an estimate of the time needed to pass through a cell
3. calculate the new particle velocity $\vec{v}_p$
4. use equation (6.21) to find the new particle position using the new particle velocity $\vec{v}_p$
5. check if the particle has left the cell

The starting time step Δt is estimated using the Courant number defined by

$$Co = \frac{U\delta t}{\delta x} \; . \tag{6.22}$$

As there are three dimensions the choose the lowest of

$$\Delta t_c = \min(\frac{Co\,\Delta x}{u_x}, \frac{Co\,\Delta y}{u_y}, \frac{Co\,\Delta z}{u_z}) \; . \tag{6.23}$$

And a value of $Co \approx 0.34$ in order to step in two or three steps through a typical cell (see an example in Figure 6.2).

6.3 Particle within a cell

One of the major tasks is to find out whether a particle or droplet is within or outside of a cell. The easiest way to find out for an arbitrary cell is to use the cell face normals (positive pointing outwards, see Figure 6.3) and a normalised position vector from the particle to the cell face centre

$$\hat{x}_{pf} = \vec{x}_{\text{cell face}} - \vec{x}_p \; . \tag{6.24}$$

Next simply calculate the dot product using the cell face normal $\vec{x}_n$

$$\hat{x}_{pf} \cdot \vec{x}_n$$

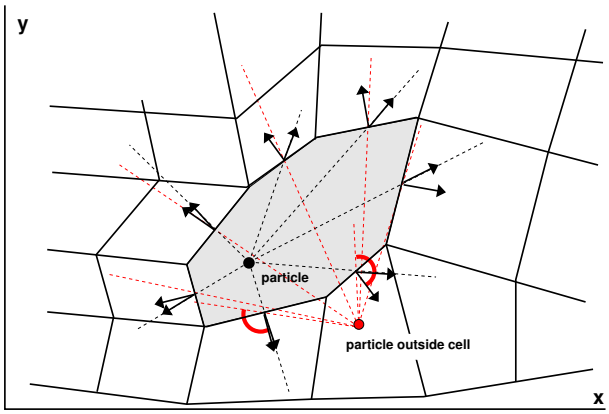


Figure 6.3: Example of particle inside and outside of a cell (red)

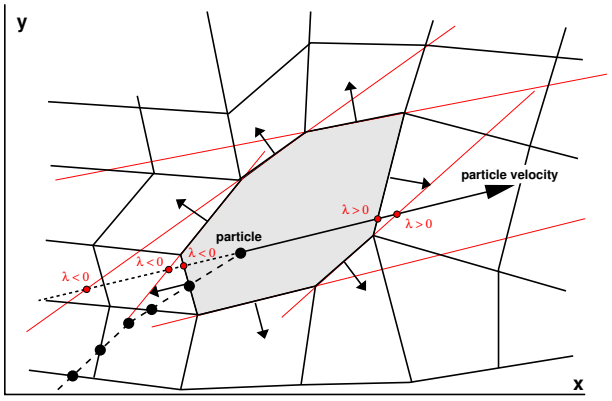


Figure 6.4: Particle intersects face

The result is

$$\begin{cases} \hat{x}_{pf} \cdot \vec{x}_n & > & 0 & \text{particle inside} \\ \hat{x}_{pf} \cdot \vec{x}_n & = & 0 & \text{particle on the face} \\ \hat{x}_{pf} \cdot \vec{x}_n & < & 0 & \text{particle outside} \end{cases}$$

Simply loop over all faces and check if the dot product does not become negative.

6.4 Intersecting a cell face

Once a particle is within a cell it will travel in some direction. It is then necessary to find out on which cell face the particle will leave the cell. When this cell face is known then the neighbouring cell is by virtue of dolfyns unstructured addressing method also known. Hence there is no need for special particle searching algorithms. Consider a particle in a cell at point $\vec{x}_p$ with a velocity $\vec{v}_p$. Any point on the line, including an intersection point $\vec{x}_i$ made with this velocity is defined as

$$\vec{x}_i = \vec{x}_p + \lambda \, \vec{v}_p \, . \tag{6.25}$$

A cell face is defined by its cell face centre and normal. For any point on the cell face, including an intersection point $\vec{x}_i$, the normal is perpendicular to the vector from the cell face centre to the intersection point. In such a case the dot product is equal to zero. Hence

$$(\vec{x}_i - \vec{x}_{\text{cell face}}) \cdot \vec{x}_n = 0 \,. \tag{6.26}$$

Insert 6.25 into 6.26 and solve for λ

$$(\vec{x}_p + \lambda\, \vec{v}_p - \vec{x}_{\text{cell face}}) \cdot \vec{x}_n = 0$$

$$(\vec{x}_p - \vec{x}_{\text{cell face}}) \cdot \vec{x}_n + \lambda\, \vec{v}_p \cdot \vec{x}_n = 0$$

or

$$\lambda = -\frac{(\vec{x}_p - \vec{x}_{\text{cell face}}) \cdot \vec{x}_n}{\vec{v}_p \cdot \vec{x}_n} \tag{6.27}$$

The lowest positive λ found is the intersection point. The point itself is then found using equation (6.25).

When the dot product $\vec{v}_p \cdot \vec{x}_n = 0$ then $\vec{v}_p$ is parallel to the face defined by $\vec{x}_n$. A special check is needed in case two, or more, faces share the same normal vector.

6.5 Interpolating velocities

The Lagrangian procedure described here requires that the local fluid velocity at some point within a cell is available. One possible method is to start directly from the cell centred velocity components and interpolate from all neighbouring cells. This path will lead to large interpolating molecules surrounding the cell the particle is currently in.

Another method is based on interpolating the velocities on to the corner nodes of the cells first (using the cell centred values and the velocity gradients) and then use these velocities as a starting point. Next the velocity inside the cell can be interpolated using for example shape functions. Such functions, however, need to take into account the cell topology in order to be able to map parameters into a computational domain. This method requires the cell topology to be used in dolfyn which is not desirable.

The currently used interpolation method is based on weighting the (interpolated) velocity components on the nodes as in

$$d = \sum_{i=1}^{N_{\text{nodes}}} W_i\, d_i \,. \tag{6.28}$$

Furthermore it is a requirement that when the interpolation point coincides with a node its value should be equal to the nodal value.

Finally the interpolation method should yield values no smaller than the minimum d_i and no larger than the maximum d_i. Thus

$$\sum_{i=1}^{N_{\text{nodes}}} W_i = 1 \text{ , with } 0 \leq W_i \leq 1 \text{ .} \tag{6.29}$$

For any point within a cell the next weighting function is used

$$W_i = \frac{(\frac{1}{r_i})^2}{\sum(\frac{1}{r_i})^2} \tag{6.30}$$

with

$$r_i = \|\vec{x}_{\text{node}} - \vec{x_p}\| \text{ .}$$

As the weight function is known the fluid velocity $\vec{u}_p$ at a particle position $\vec{x}_p$ is then simply

$$\vec{u}_p = \sum_{i=1}^{N_{\text{nodes}}} W_i \, \vec{u}_{\text{node}_i} \text{ .} \tag{6.31}$$

6.6 Examples

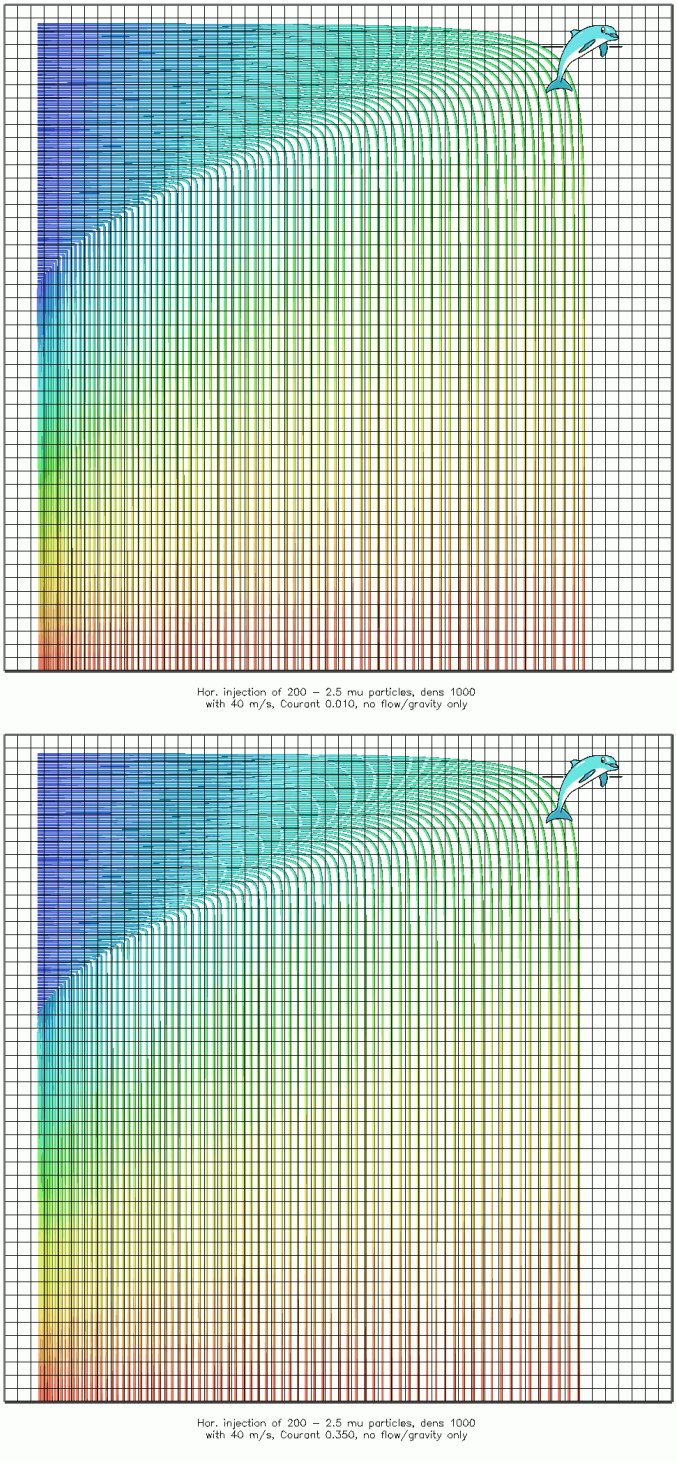


Figure 6.5: Free falling particles, influence of particle Courant number

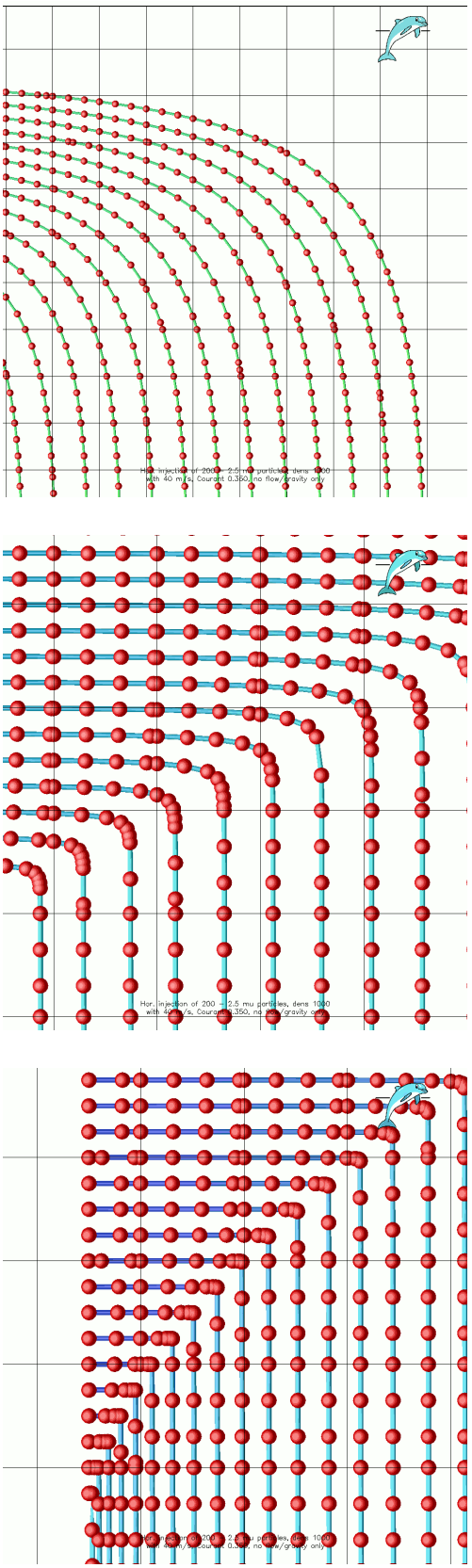


Figure 6.6: Free falling particles, details showing steps within cells

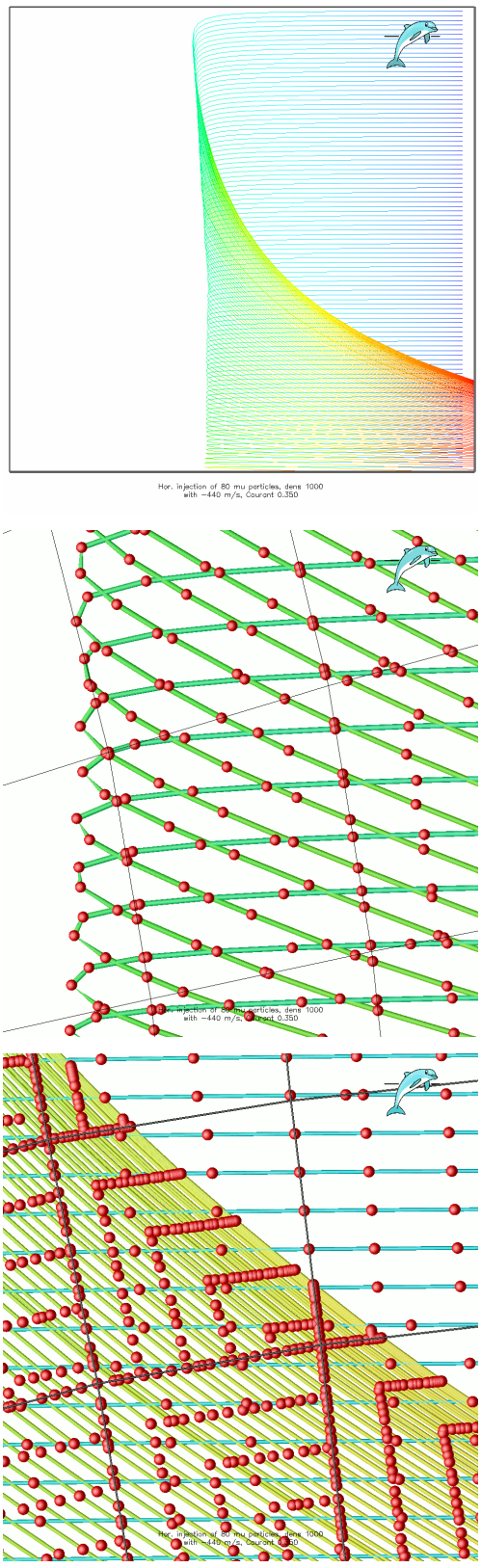


Figure 6.7: Particles in a stagnation flow

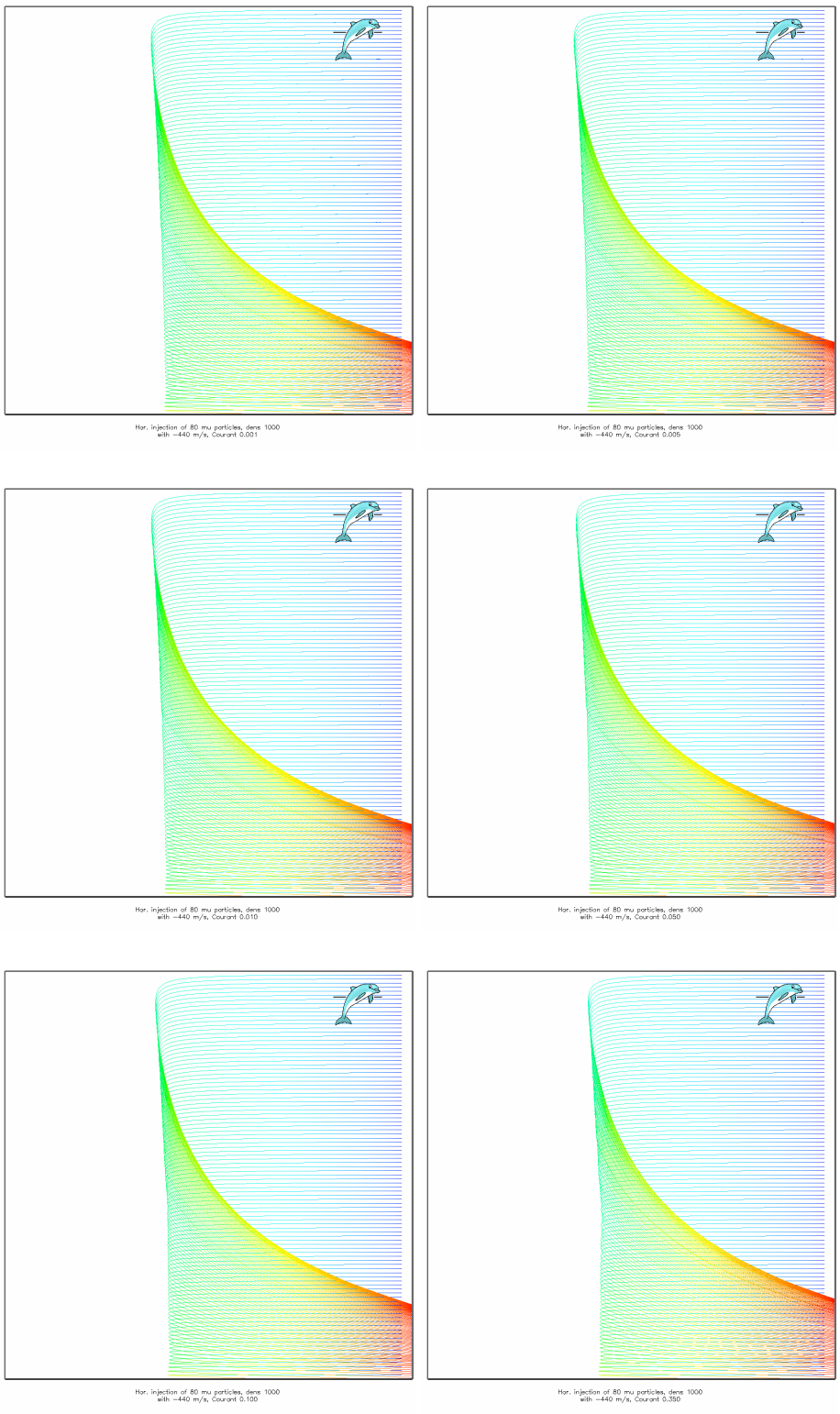


Figure 6.8: Particles in a stagnation flow, influence of particle Courant number

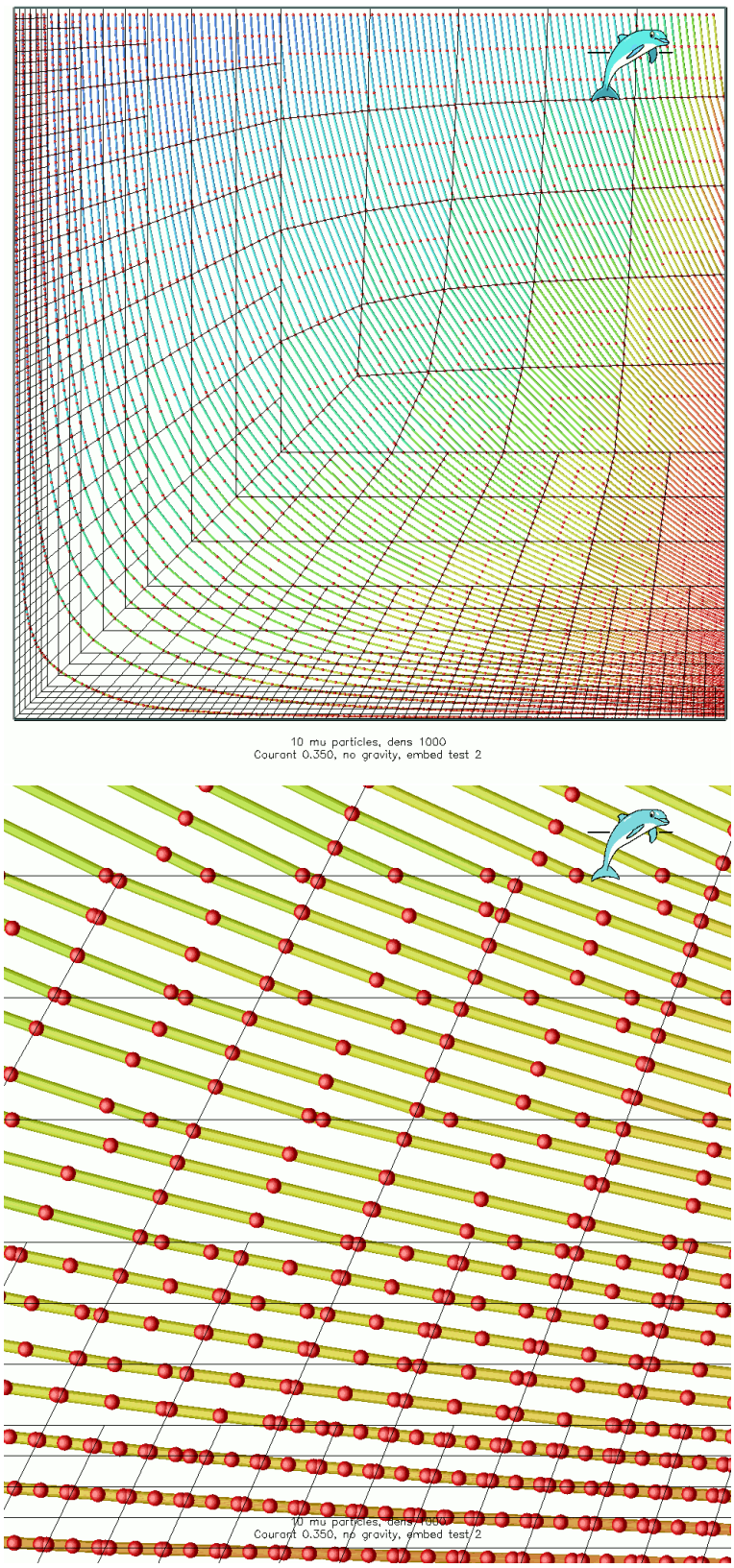


Figure 6.9: Small particles/streamlines in a stagnation flow

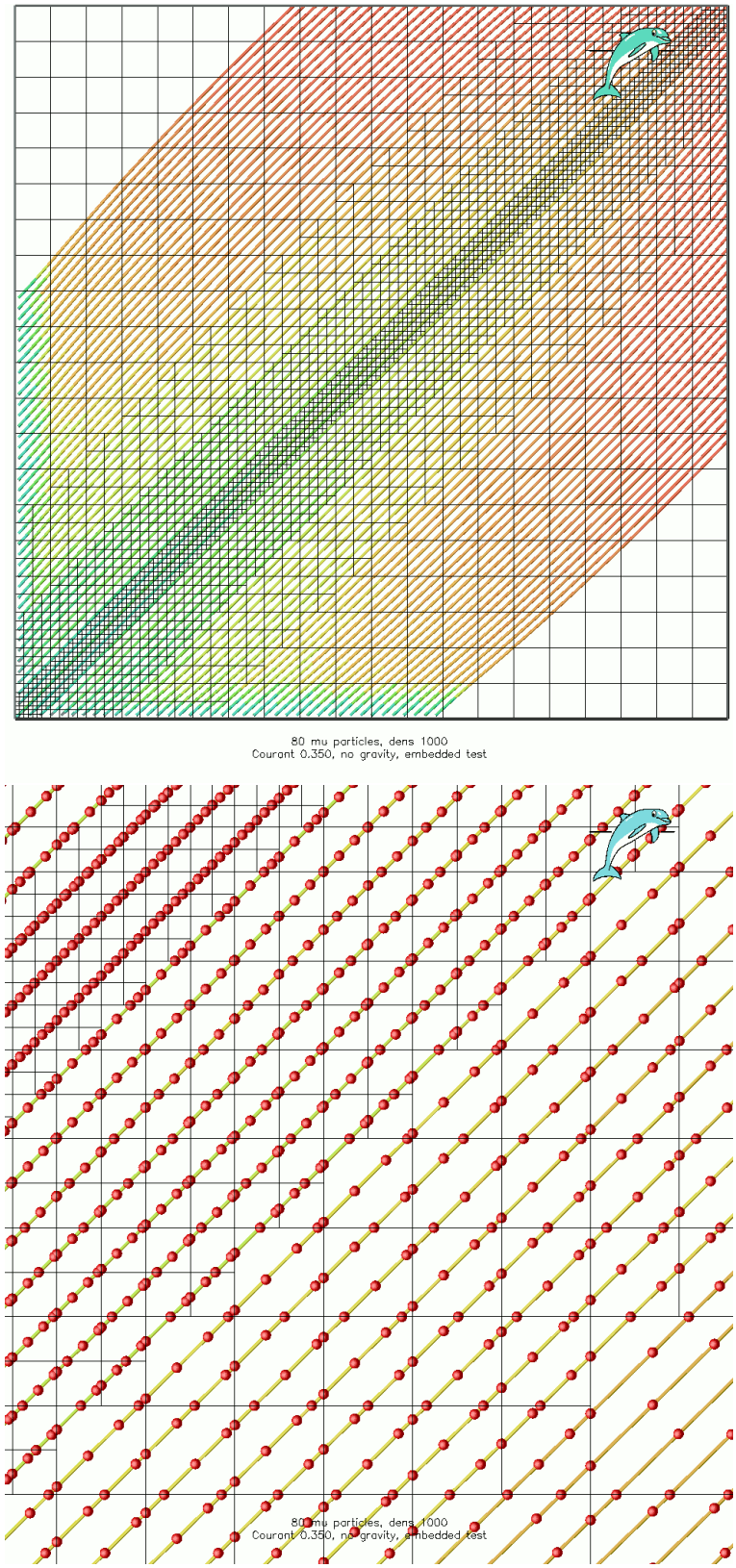


Figure 6.10: Small particles/streamlines in an 45 angle test

Appendices

A Geometry input files

You will need three geometry files *casename*.cel, *casename*.vrt, and *casename*.bnd, where *casename* as the name of your case. The preprocessor only needs these simple files:

- A file with cell definitions (*.cel)
- A file with the vertices (*.vrt)
- And the boundaries (*.bnd)

The results are stored in *casename*.geo.

The definitions of the cells in *.cel are:

```
nr , i1 , i2 , i3 , i4 , i5 , i6 , i7 , i8 , id
```

With the cell number ‘nr’, and the 8 corner nodes of a hexaeder (i1 t/m i8). Each cell has a pointer to an ‘id’ in a table. This ‘id’ is reserved for the future (for the time being set it to ‘1’).

The accompanying ‘format statement’ is a ‘free format’ (*).

Cell shapes are defined as (see Figure A.1):

Hexaeder i1, i2, i3, i4, i5, i6, i7, i8

Prism or wedge i1, i2, i3, i3, i5, i6, i7, i7

Pyramid i1, i2, i3, i4, i5, i5, i5, i5

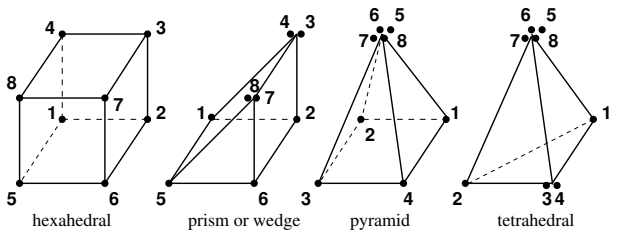


Figure A.1: Cell definitions

Tetraeder i1, i2, i3, i3, i5, i5, i5, i5

The vertices in *.vrt are:

```
nr, x, y, z
```

With the number ‘nr’ and it’s position in 3D (x, y, z). The accompanying ‘format statement’ is: *i9,6x,3g16.9*.

Finally the boundaries in *.bnd are:

```
nr, i1, i2, i3, i4, id
```

With number ‘nr’ and the 4 corner nodes of a face of a hexaeder (i1 t/m i4). The boundaries point to an ‘id’ which values will be set in the control file (*.din). All boundary faces without an ‘id’ will be given the *default id of ‘0’*.

The accompanying ’format statement’ is in ’free format’ (*). Two variants are defined:

Quad i1, i2, i3, i4

Tri i1, i2, i3, i3

B Verification with equidistant Lid Driven Cavity tests

Following Roache [35] an error analysis has been performed using 'Richardson Extrapolation' on various meshes and convective differencing schemes. All the meshes are simple orthogonal and equidistant. Although dolfyn contains non-orthogonal corrections etcetera, still this is a valuable test because all additions should not break the basic fundament of the code. The reference data is by Ghia, Ghia, and Shin [15].

The cavity has dimensions of one by one and the lid has a velocity of one, leaving the (laminar) viscosity of the fluid with density one to determine the Reynolds number ($Re = \rho V_{\text{lid}} L_{\text{lid}}/\mu$). The calculations were done with Gauss for the gradients and various convective differencing schemes:

UD Classic standard first order Upwind Differencing.

CD 0.8 Blend of 80% second order Central Differencing and 20% first order upwind differencing.

CD1 Pure second order Central Differencing.

LUD Second order Linear Upwind Differencing with a Convection Bounded Criterion (UD outside the Normalised Variable range of $0 \leq \tilde{\Phi}_C \leq 1$).

Gamma The Gamma CBC differencing scheme (blending with CD).

MinMod The MinMod CBC differencing scheme (based on LUD and CD).

LUX Pure LUD not based on NVD..

Shown are the u component velocity profiles half way the cavity ($x = 0.5$, $0 \leq y \leq 1$) for all the schemes at the finest mesh (128x128) and UD and LUX only as a function of the mesh (8x8, 16x16, 32x32, 64x64, 128x128). And the development of the interpolated u component at $x = 0.5, y = 0.5$. The latter value has been linearly interpolated (which might have some effects on the final results) and is shown as a function of mesh size h (linear) and h^2 (quadratic). A second order method will have to show up as a straight line at the smallest meshes.

Also shown are streamlines based on the LUX data like in Figure B.1. They were made with OpenDX after the cell-centered velocties have been interpolated to the nodes and creating streamlines (the 'post' and 'streamline' modules of

OpenDX). Because of this procedure the streamlines will lose some of their accuracy near the walls. However nice closed streamlines do show up in the center of the main vortex.

In all this study is the result of at least 300 runs.

Brief discussion of the results:

Re 25 A very very viscous flow. The flow is dominated by the (second order) viscous forces. All convective schemes coincide on the largest mesh (128x128). The first order behaviour of UD is clearly visible as well as the second order nature of CD1 and LUX (note that CD 0.8 lays nicely in between of CD1 and UD).

Re 100 The first case were data is available from Ghia et.al. As can be seen LUX on at a mesh of 32x32, or even 16x16, already produces the final result. The Richardson Extrapolation curves clearly supports this. Note the start of the two lower corner vortices.

Re 400 A 16 times lower viscosity compared to Re 25 starts to show some differences especially with UD; a good result is only possible at the finest 128x128 mesh. The second order schemes CD1 and LUX reproduce the reference data exactly, closely followed by CD 0.8. The LUX results are already there on the 64x64 mesh (but the 32x32 are not bad either). The lower right corner vortex increases.

Re 1,000 Basically the same results as for Re 400. In the top left the vortex is about to appear.

Re 3,200 Now the differences between the schemes get very clear. Note that the NVD blended schemes follow the UD scheme whereas the two unbounded second order schemes provide the best result (again followed by the CD/UD blend). The LUX scheme is the best and the Richardson Extrapolation curve of h^2 shows a 'tail' for the CD1 scheme. The latter is now starting to deteriorate.

Re 5,000 Now only LUX is the only one which follows the data, and only on the finest 128x128 mesh. The 'tail' of CD1 in the h^2 Richardson Extrapolation curve is more pronounced. In the paper by Ghia et.al the results are show on a 257x257 mesh and a second vortex starts to appear in the lower left corner (see Figure B.3).

Re 7,500 At Reynolds 7,500 and on a mesh of 128x128 even LUX is not able to reproduce the reference data. Also instabilities start to occur; see Figure B.10.

Re 10,000 The results of Reynolds 10,000 resemble the Reynolds 7,500 results. The results in Figure B.11 are based on a 256x256 mesh and Figure B.12 shows the corresponding streamlines. It is clearly visible that the upwind and CBC upwinded schemes are the most stable. The (unboudded) second order schemes CD1 and LUX show unsteady effects; the latter is also visisble in the residual drop which do not reach machine accuracy

levels anymore. The results in the reference paper are based on a high order upwind scheme with a larger stencil (with a face based unstructured solver one has to restrict to a small stencil). Nevertehless at some point instabilities as can be seen in Figure B.12 will have to pop up at some point.

General conlusion is that dolfyn is a second order accurate code which provides for this particular case and mesh topology very accurate and correct results. Also for testing purposes the lid driven cavity can be used at medium Reynolds numbers; for example only at Re 400 or the combination Re 100 and Re 1,000.

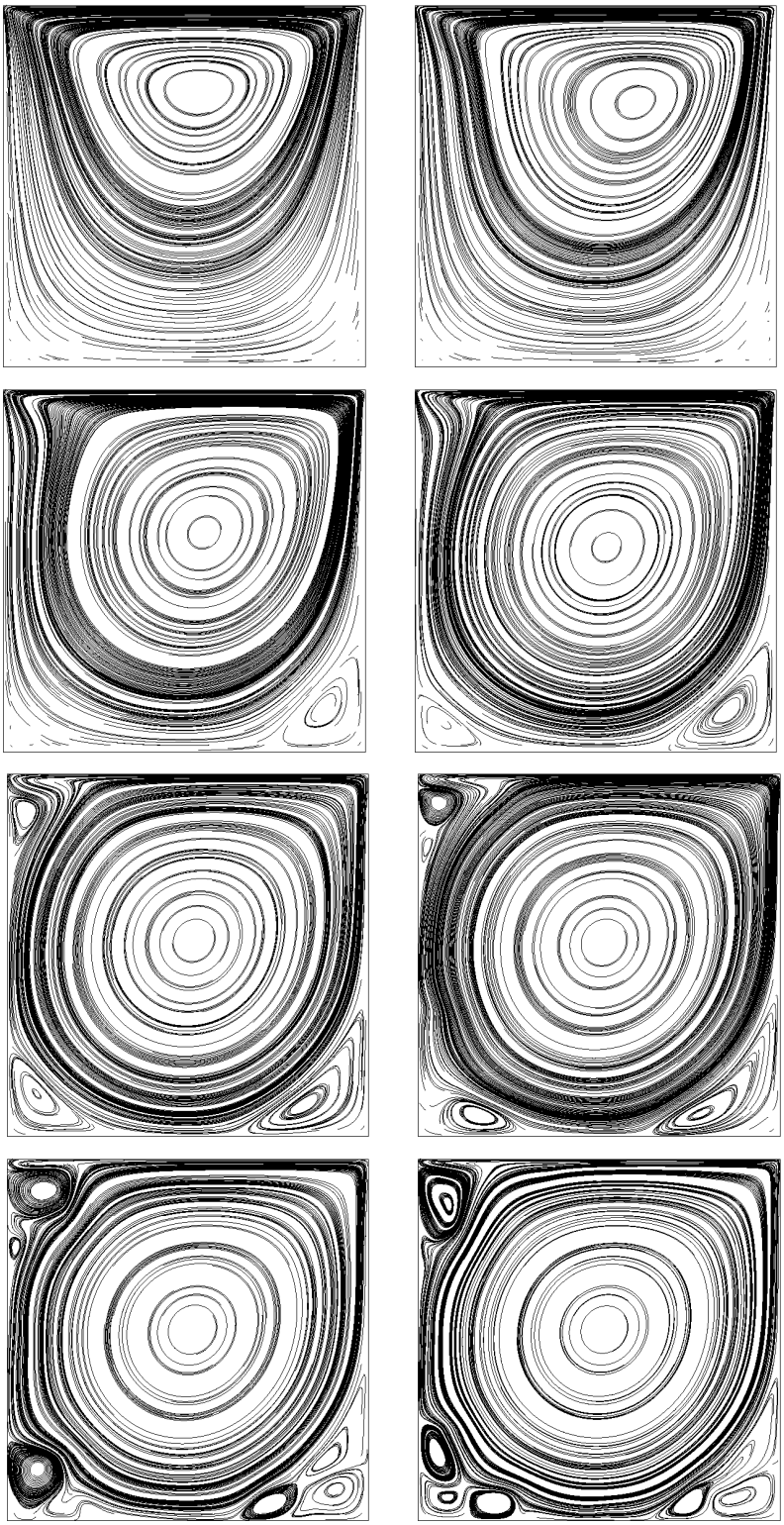


Figure B.1: Streamlines at Reynolds 25, 100, 400, 1,000, 3,200, 5,000, 7,500 and 10,000 with 128x128 and LUX

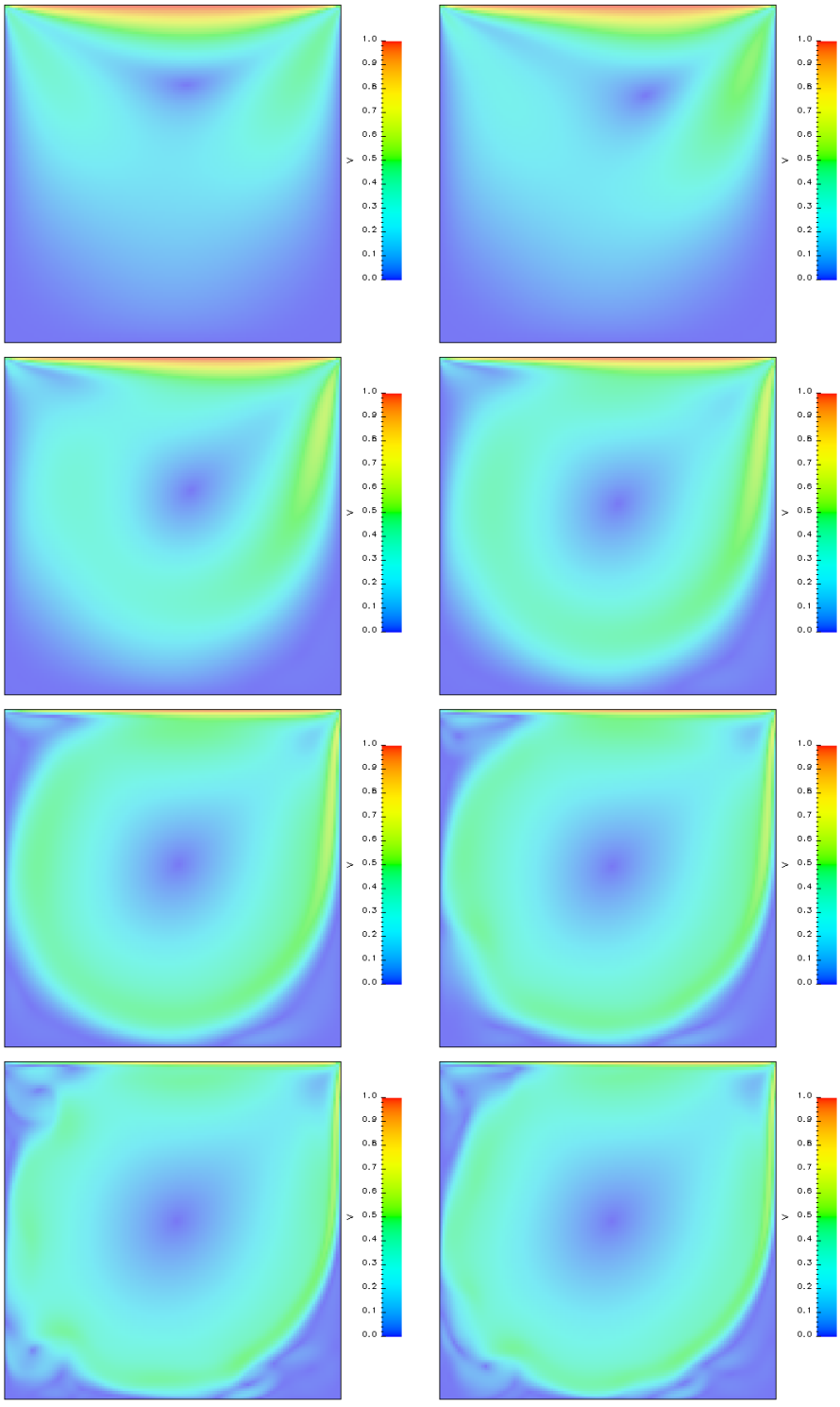


Figure B.2: Velocity magnitude at Reynolds 25, 100, 400, 1,000, 3,200, 5,000, 7,500 and 10,000 with 128x128 and LUX

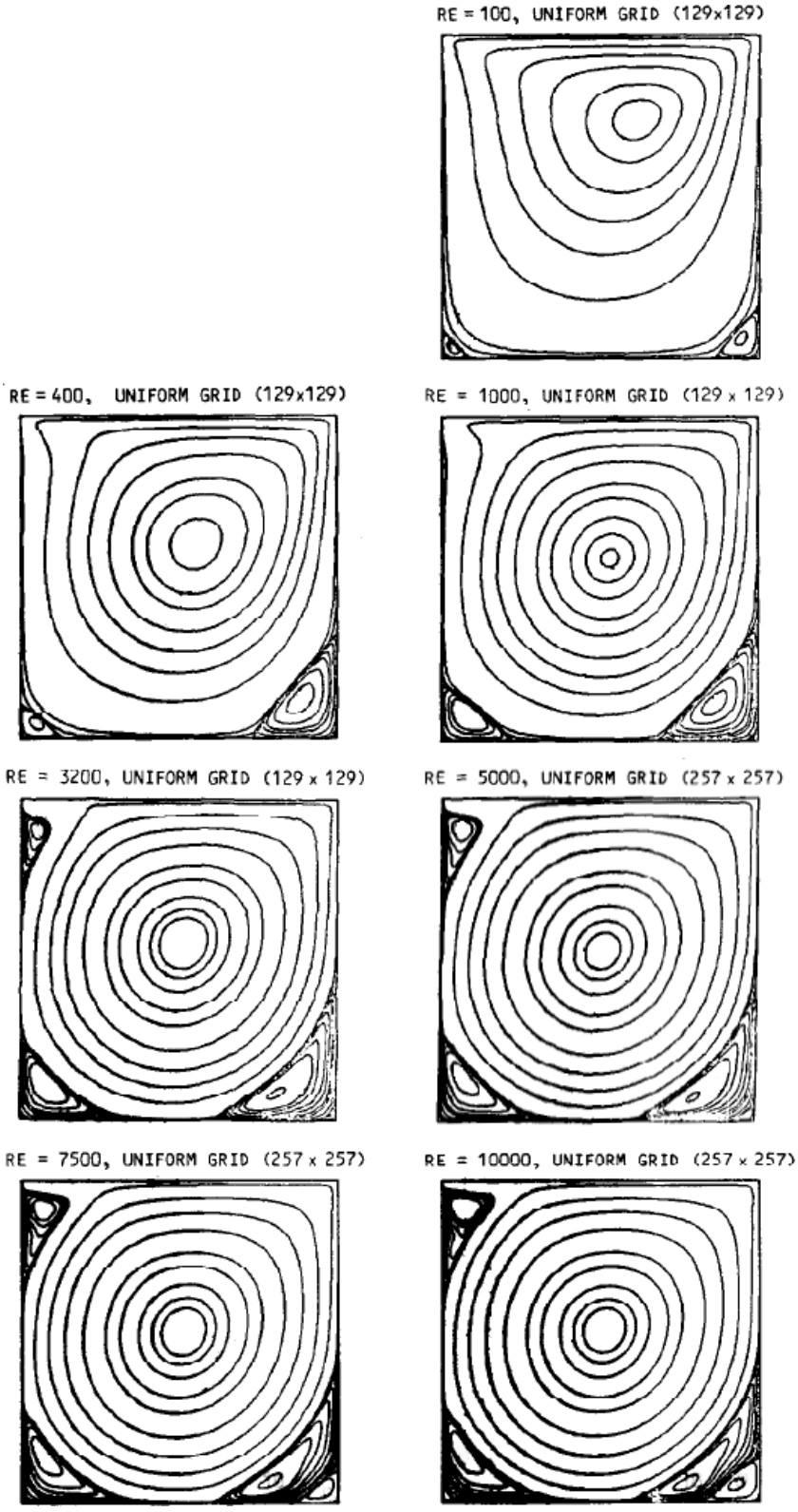
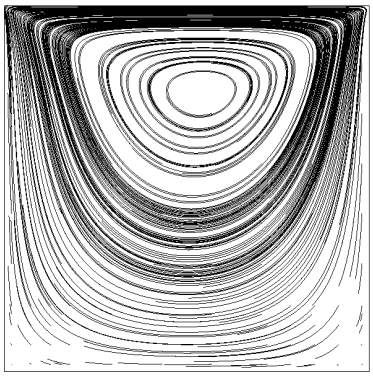
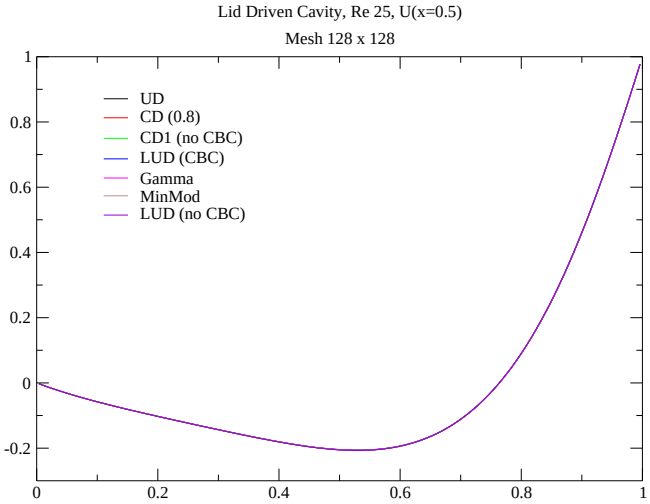
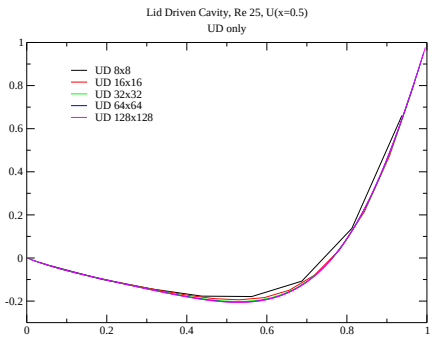
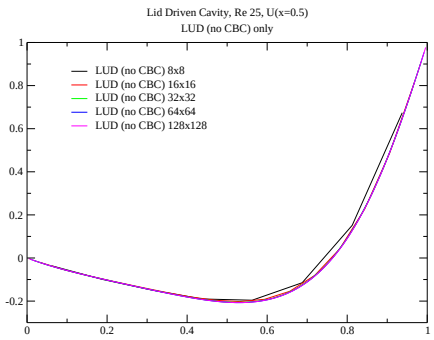


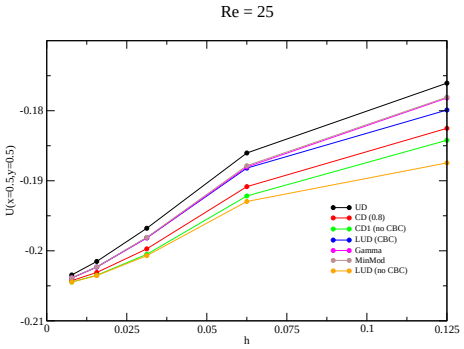
Figure B.3: Results from Ghia et.al











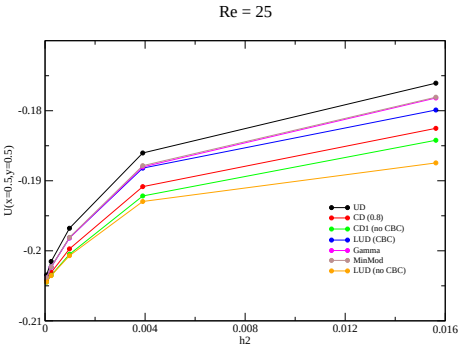


Figure B.4: Reynolds 25

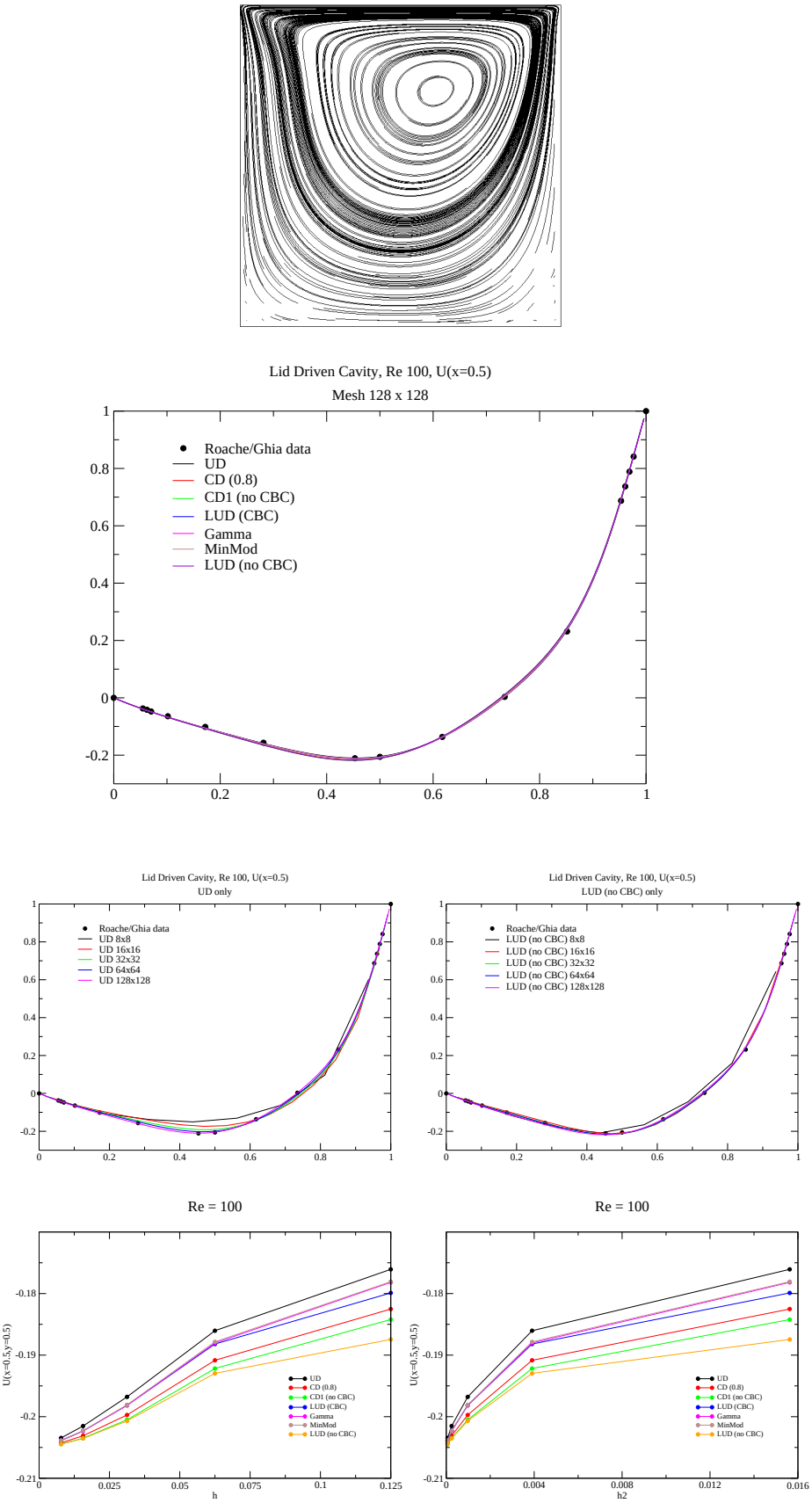
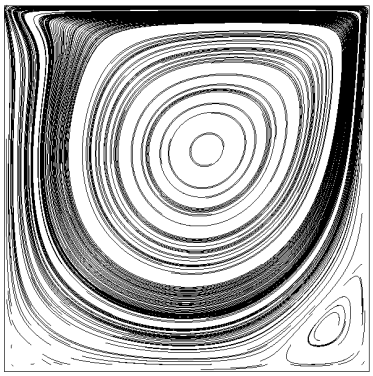
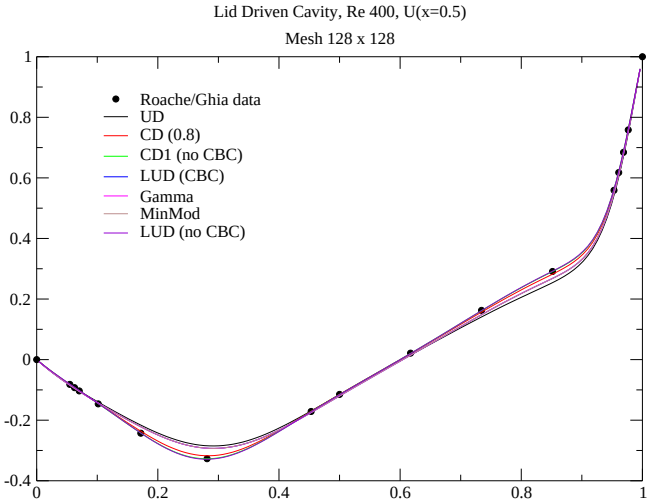
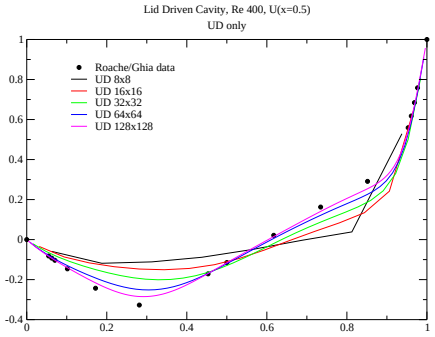
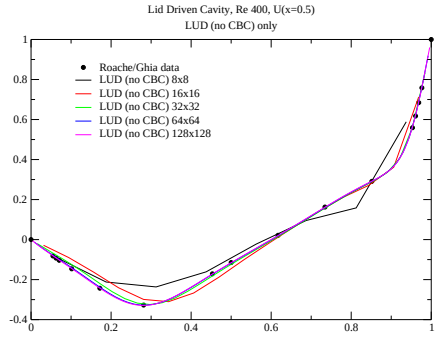


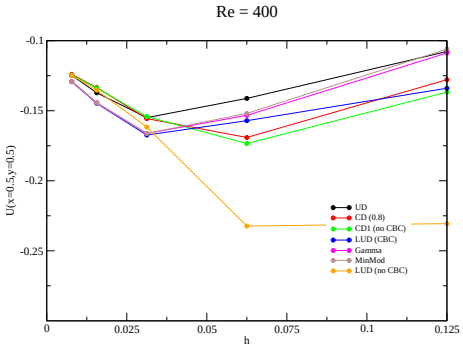
Figure B.5: Reynolds 100











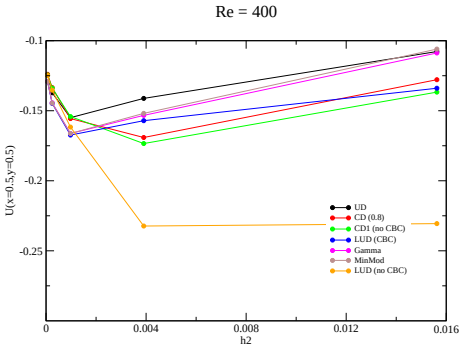
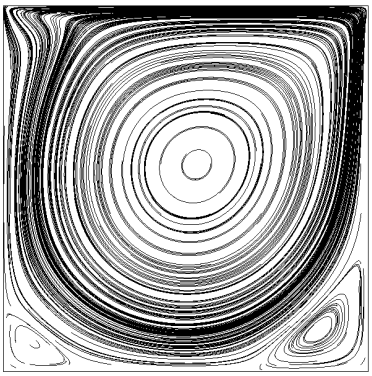
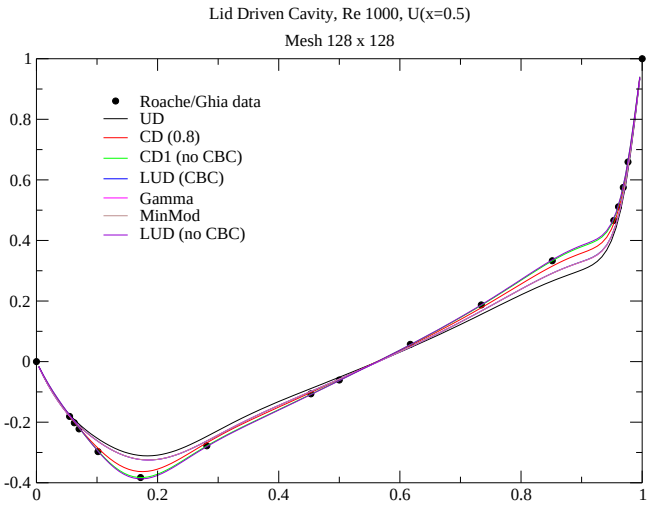


Figure B.6: Reynolds 400





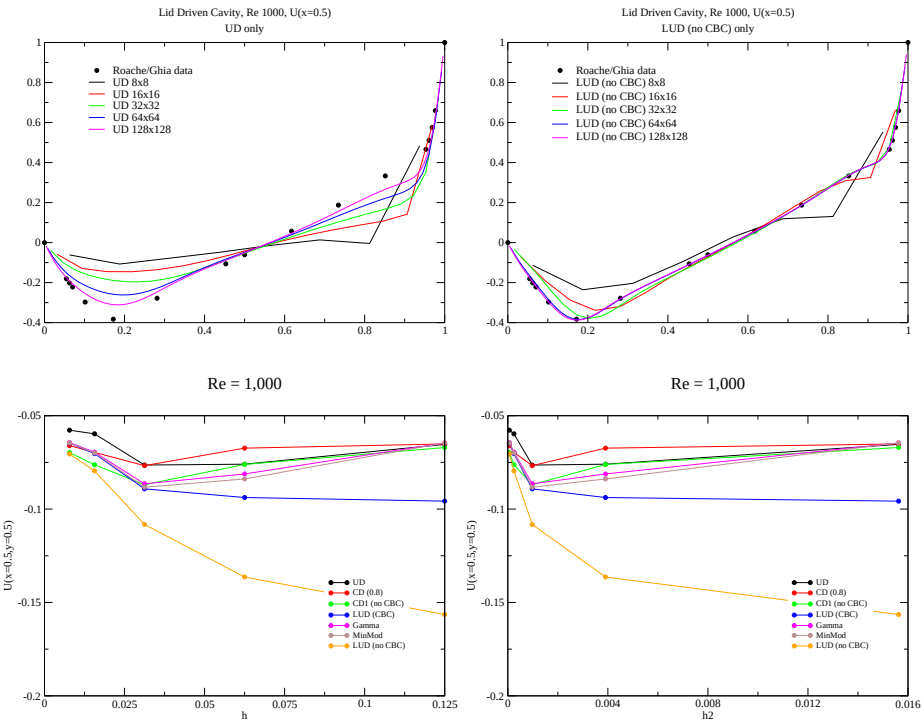
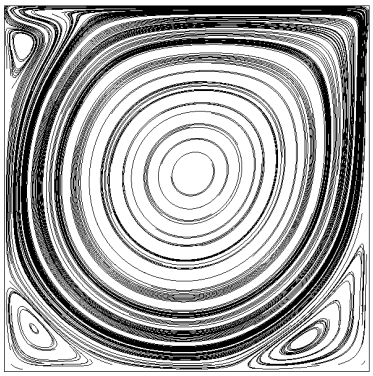
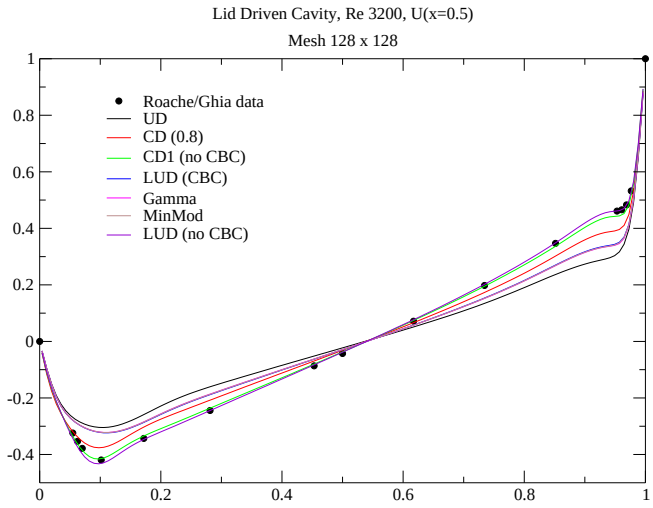
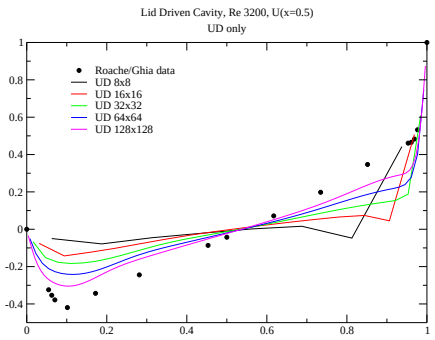
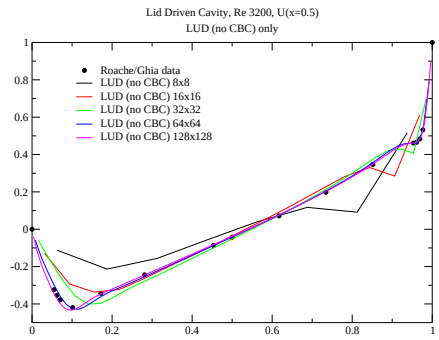


Figure B.7: Reynolds 1,000









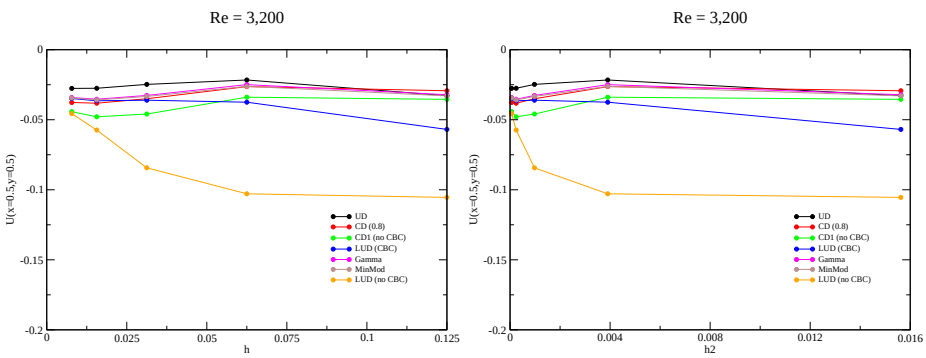
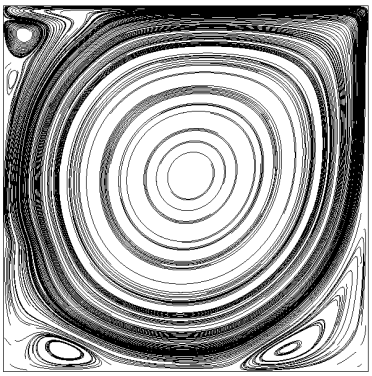
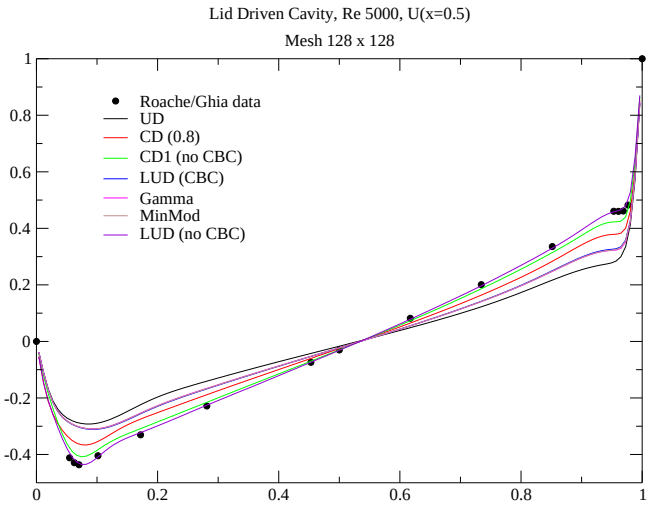


Figure B.8: Reynolds 3,200





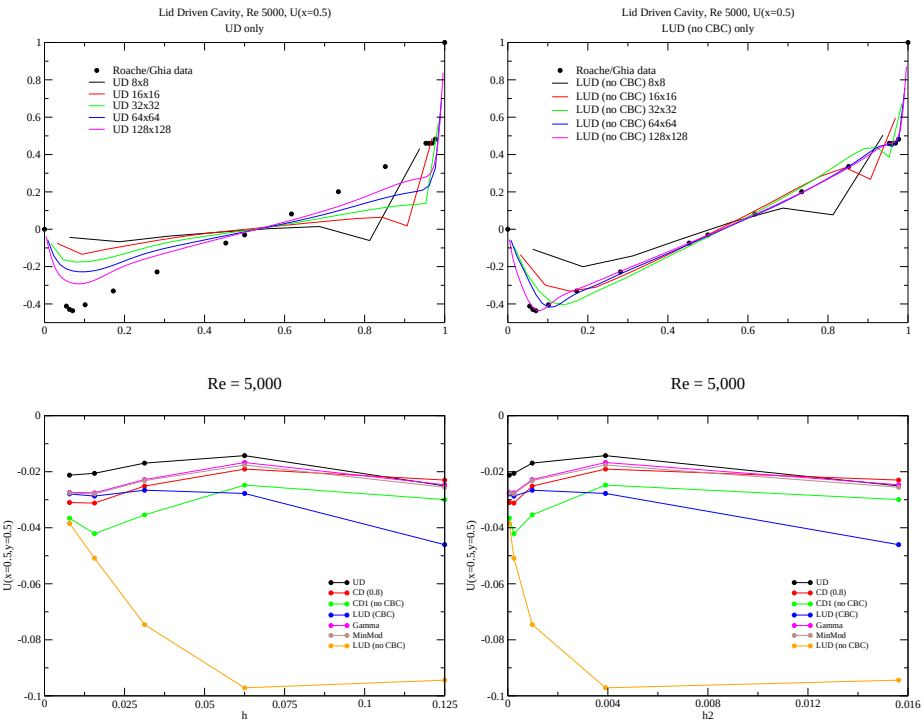
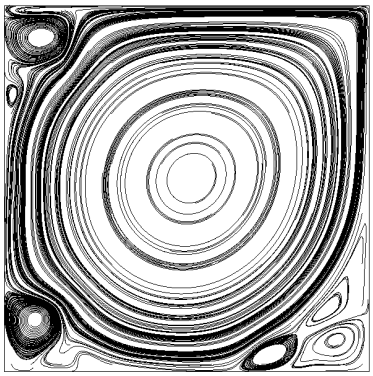
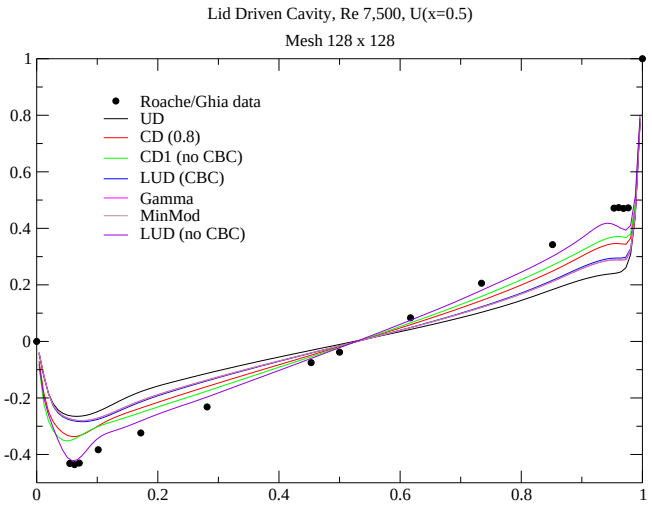


Figure B.9: Reynolds 5,000





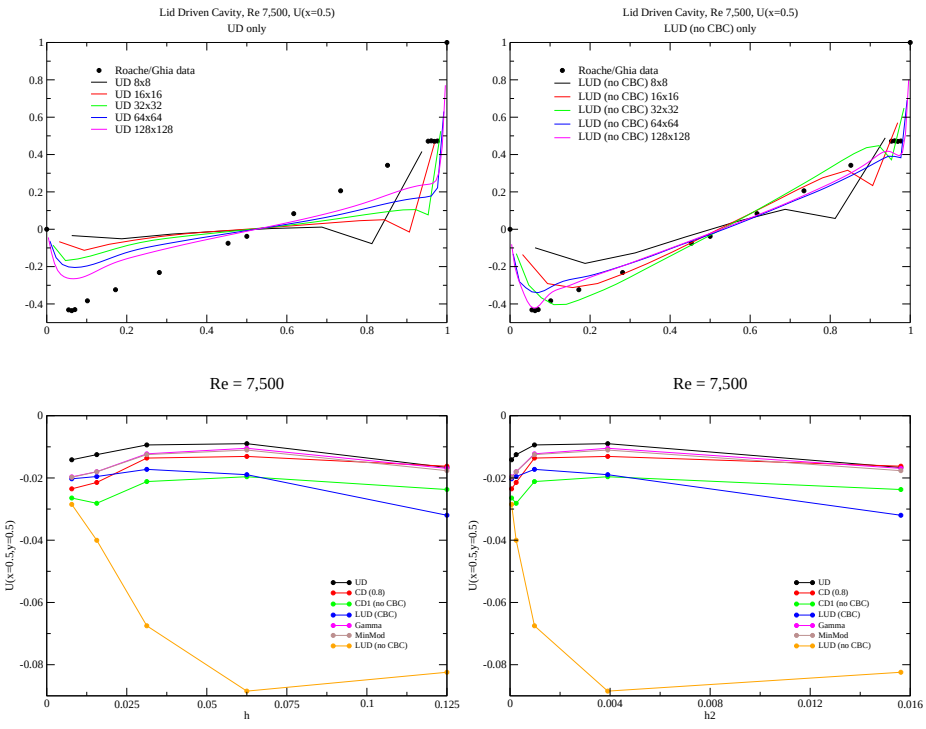
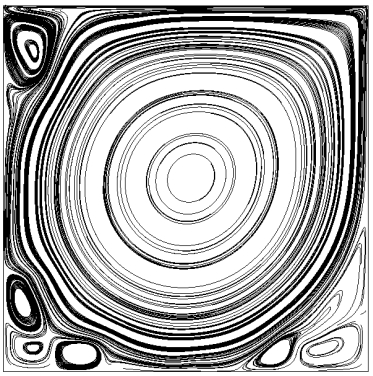
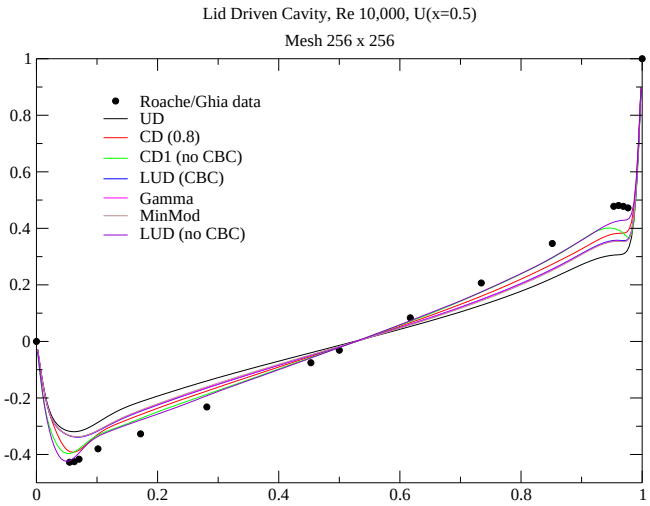


Figure B.10: Reynolds 7,500





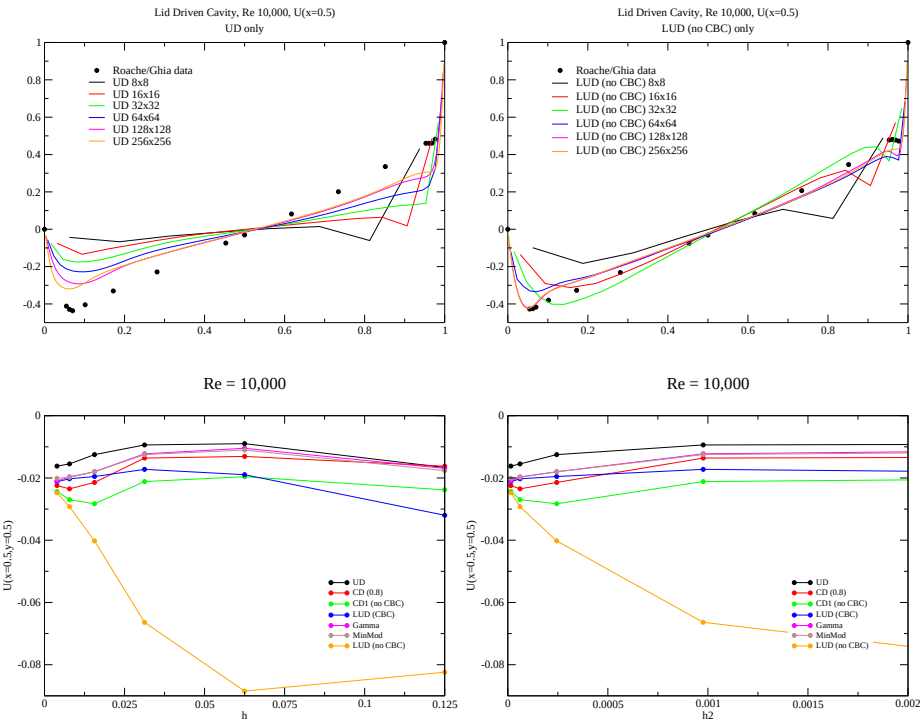


Figure B.11: Reynolds 10,000

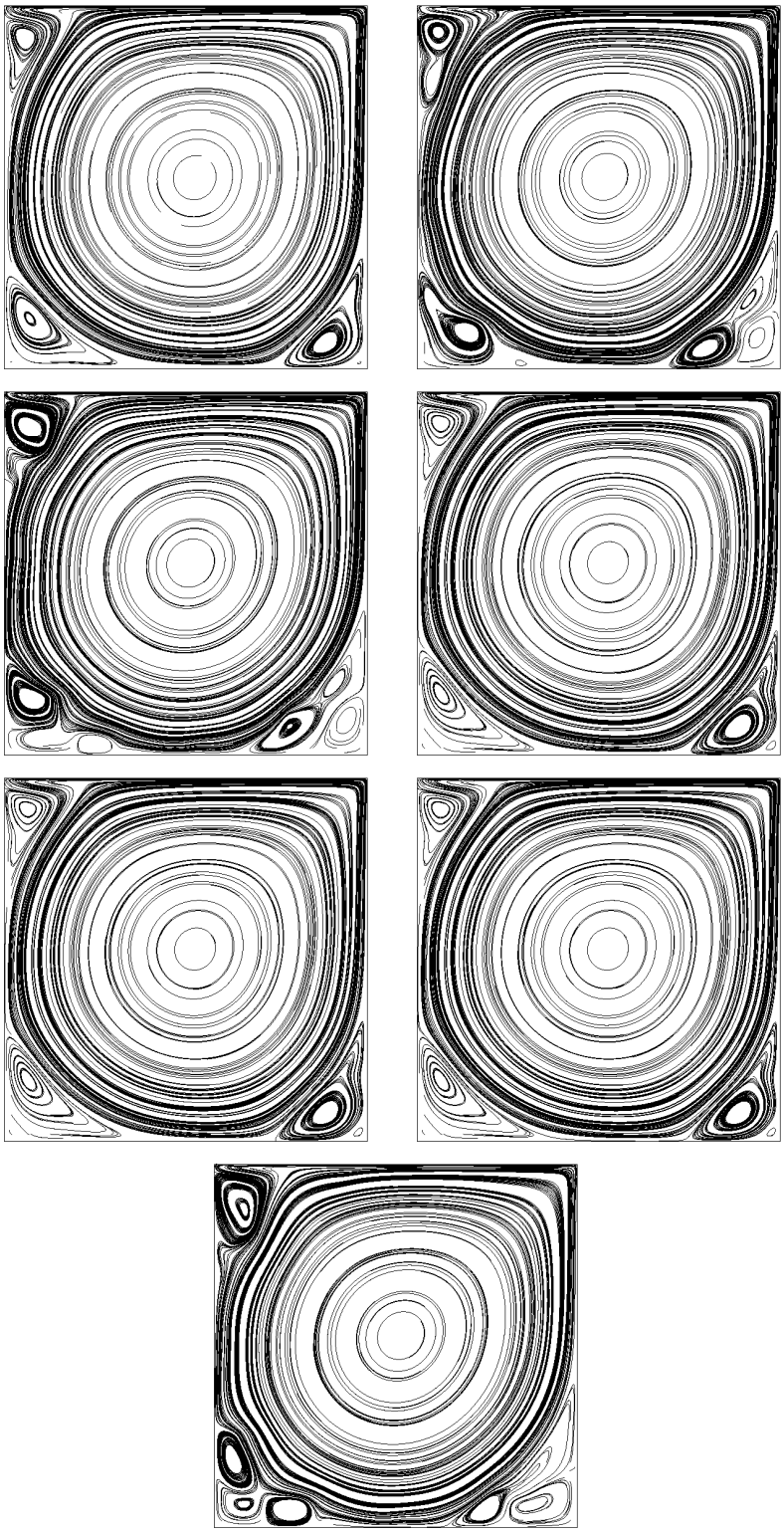


Figure B.12: Reynolds 10,000 at 256x256 and influence of differencing scheme (UD, CD, CD1, LUD, Gamma, MinMod, LUX)

C Various tests

C.1 C1 Unstructured

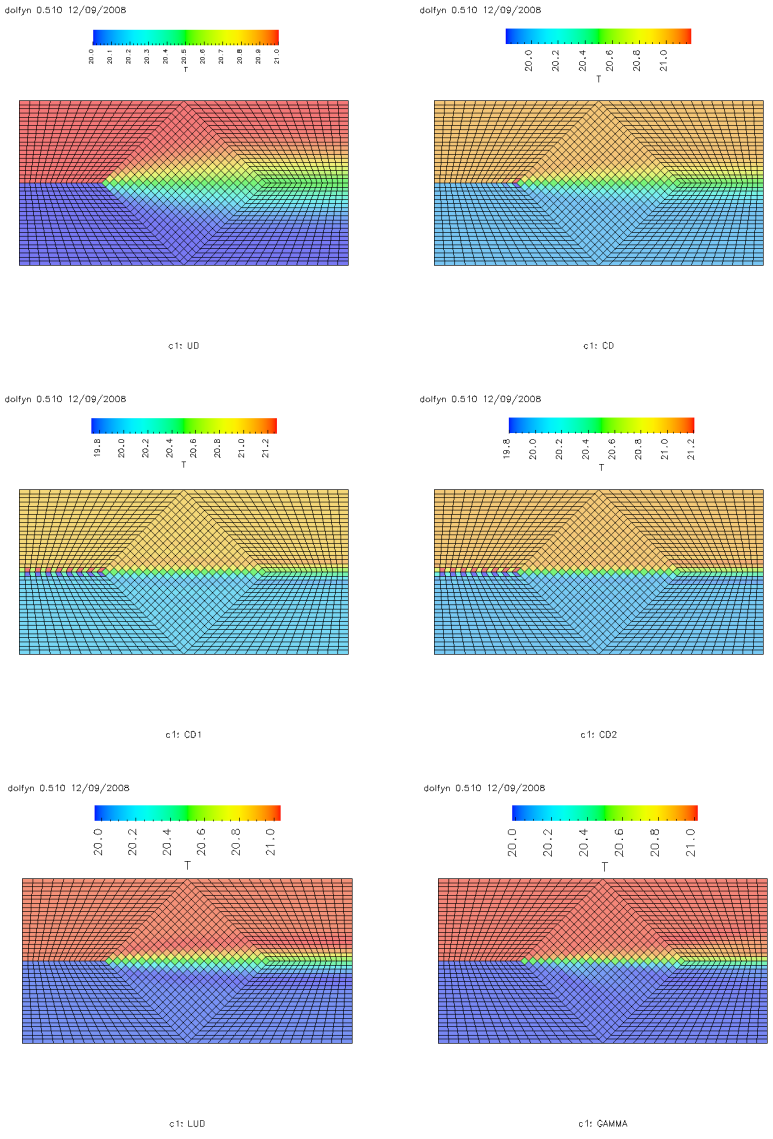


Figure C.1: C1 UD, CD(0.8), CD1, CD2, LUD, Gamma

C.2 C3 Leonard tests

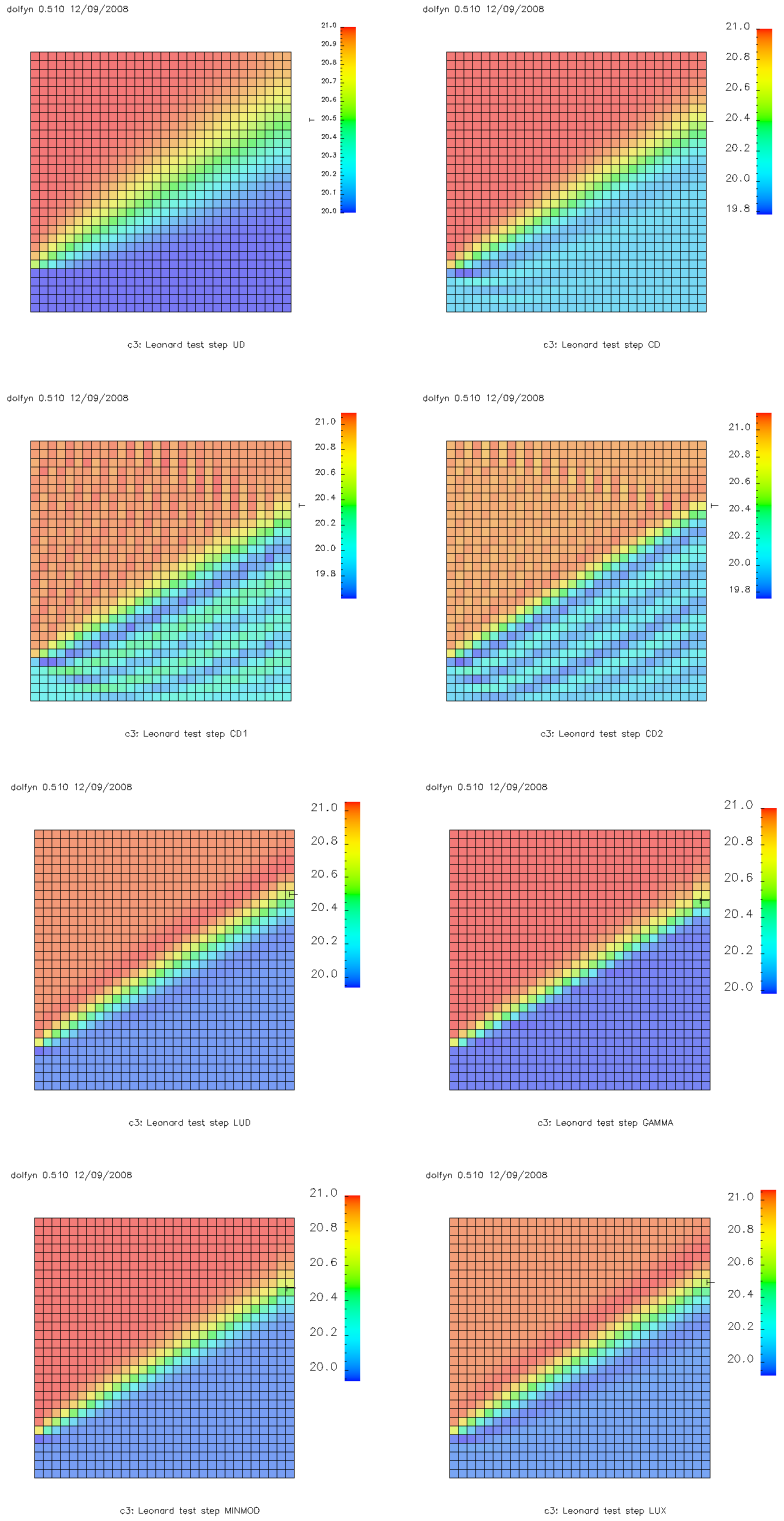


Figure C.2: Leonard step UD, CD(0.8), CD1, CD2, LUD, Gamma, MinMod, LUX

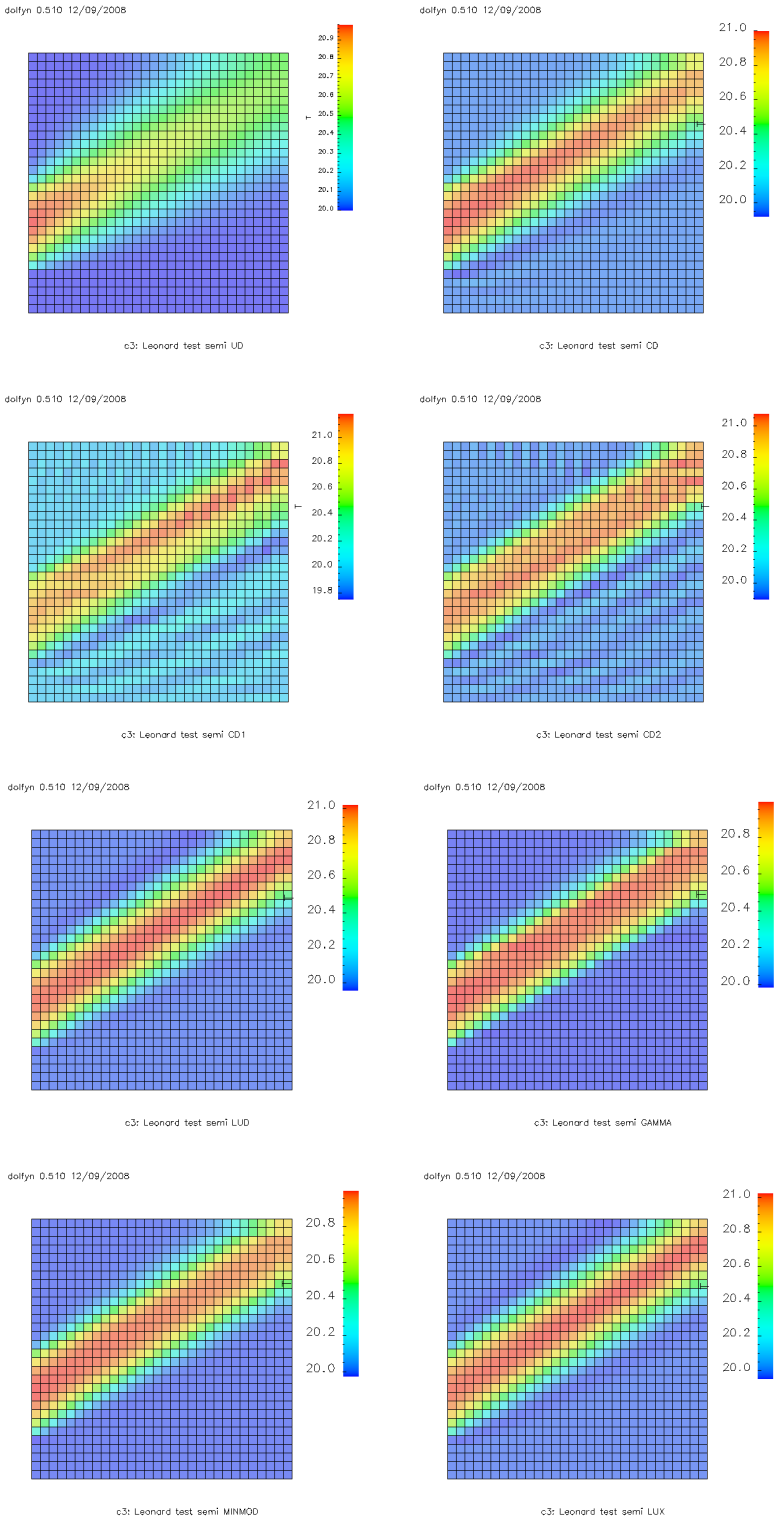


Figure C.3: Leonard semi UD, CD(0.8), CD1, CD2, LUD, Gamma, MinMod, LUX

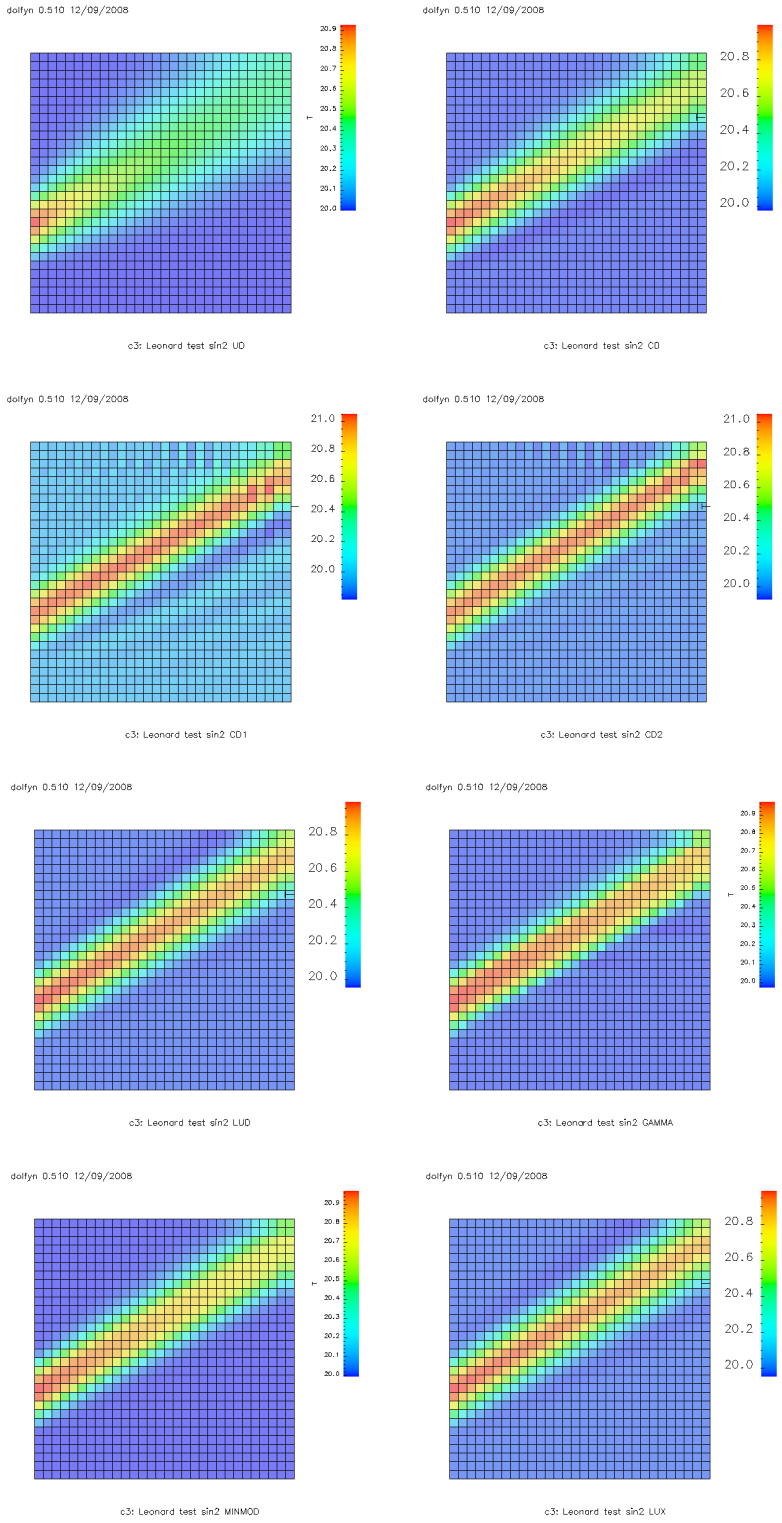


Figure C.4: Leonard sin2 UD, CD(0.8), CD1, CD2, LUD, Gamma, MinMod, LUX

C.3 C6 Wedges

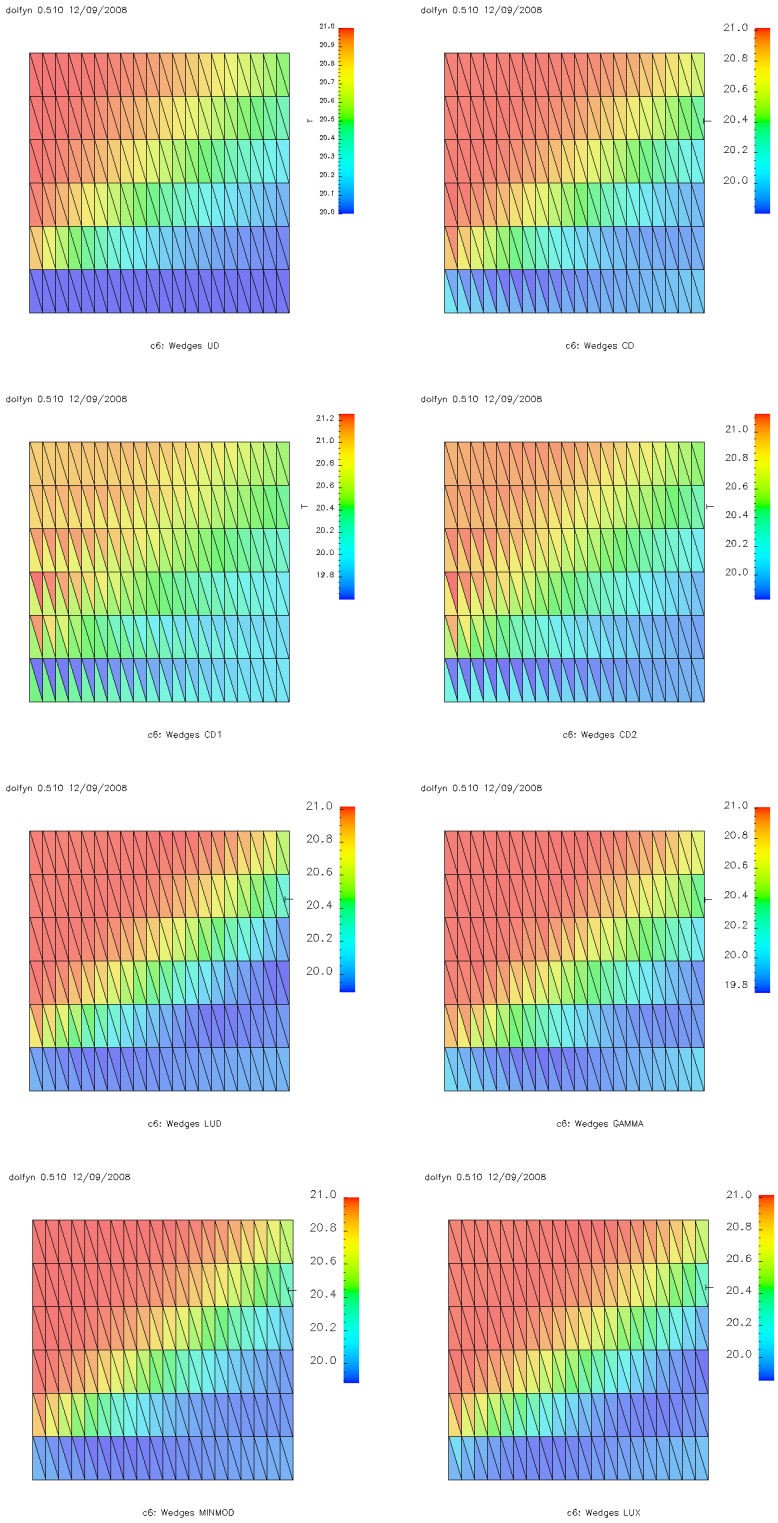


Figure C.5: C6 Wedges UD, CD(0.8), CD1, CD2, LUD, Gamma, MinMod, LUX

C.4 C7 Mesh jump

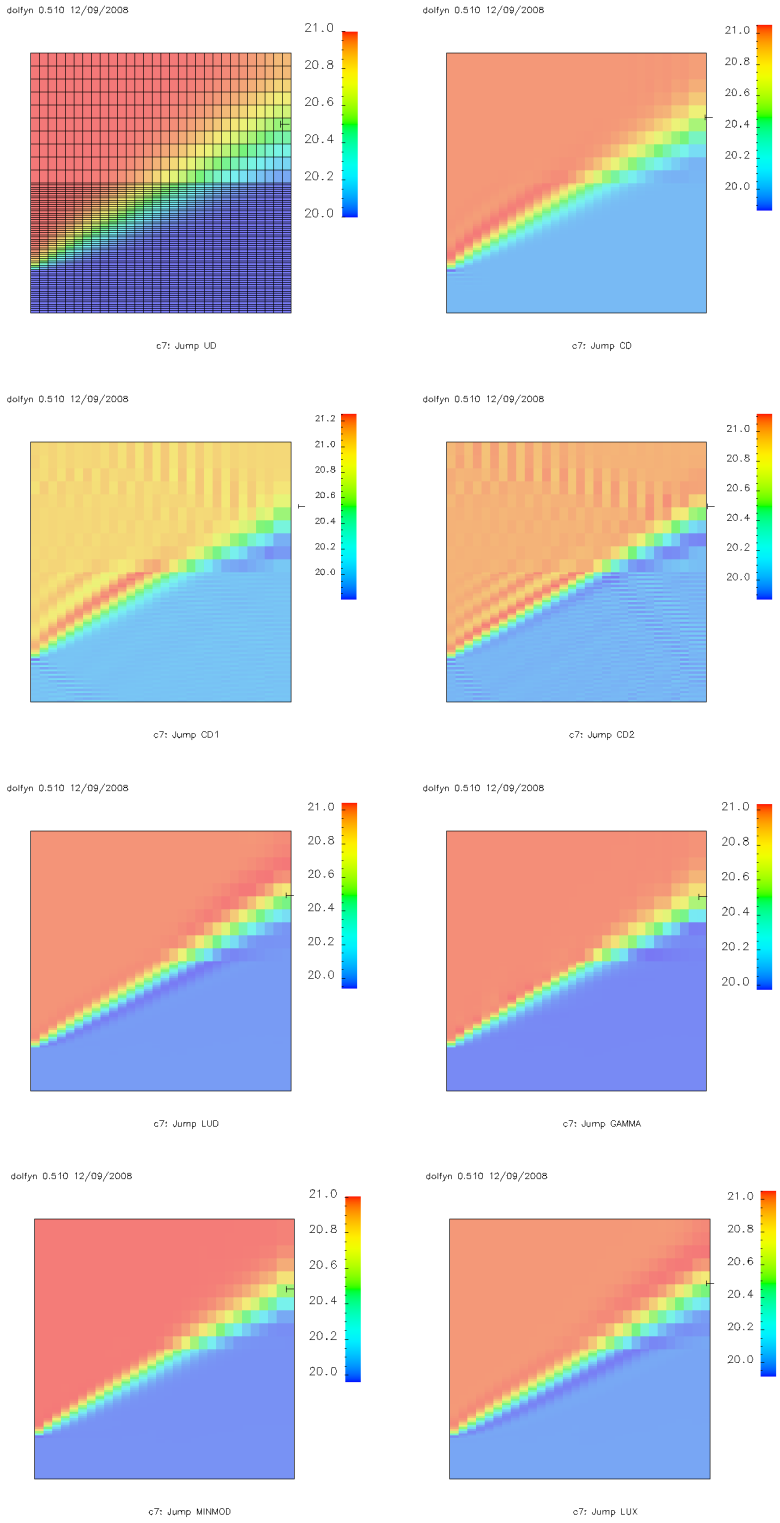


Figure C.6: C7 Jump UD, CD(0.8), CD1, CD2, LUD, Gamma, MinMod, LUX

C.5 45 degrees lid driven cavity

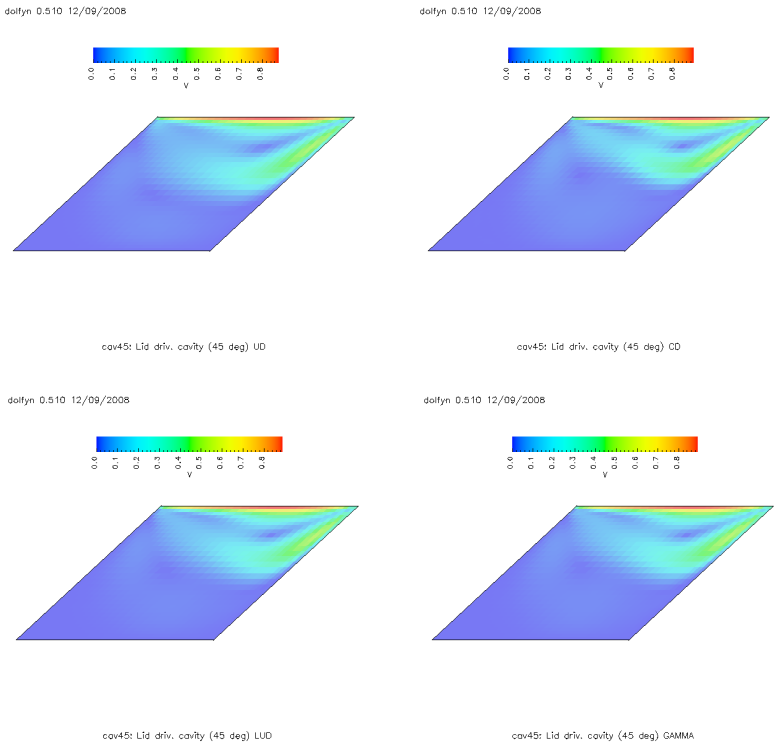


Figure C.7: 45 degrees UD, CD(0.8), LUD, Gamma

C.6 Lid driven cavity with tetrahedral cells only I

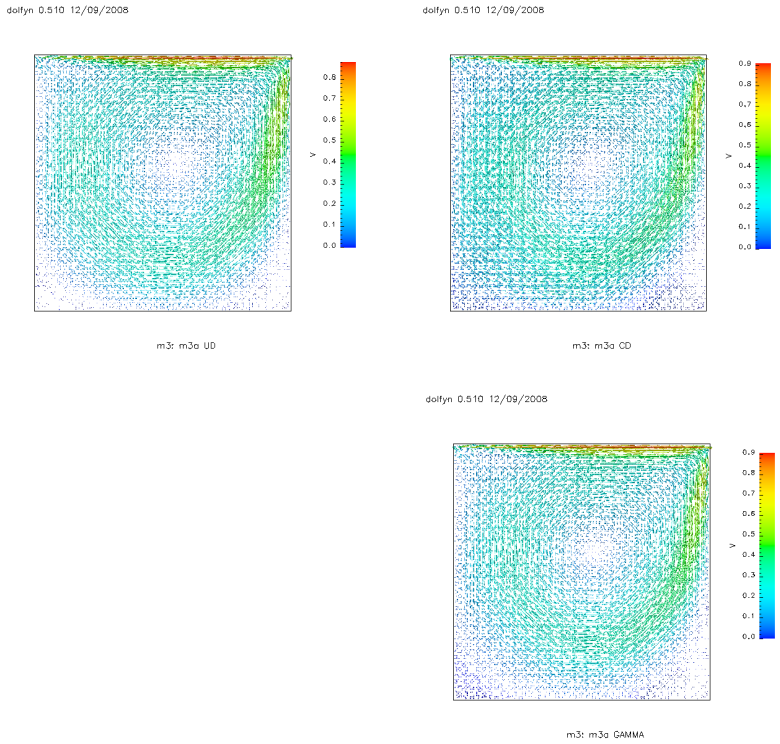


Figure C.8: m3a lid driven cavity, UD, CD(0.8), LUD, Gamma

C.7 Lid driven cavity with tetrahedral cells only II

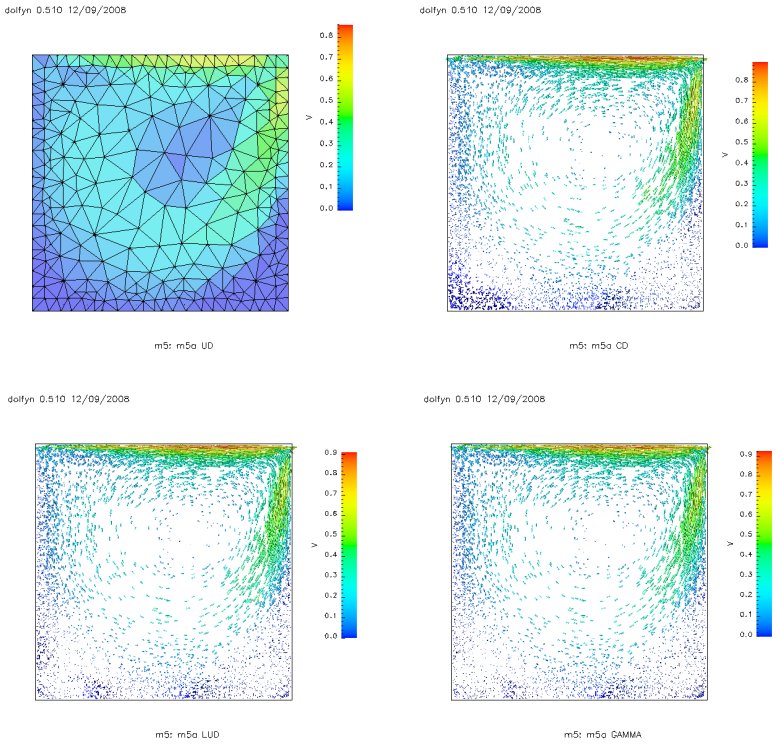


Figure C.9: m5a lid driven cavity, UD, CD(0.8), LUD, Gamma

C.8 Walls with tetrahedral cells only

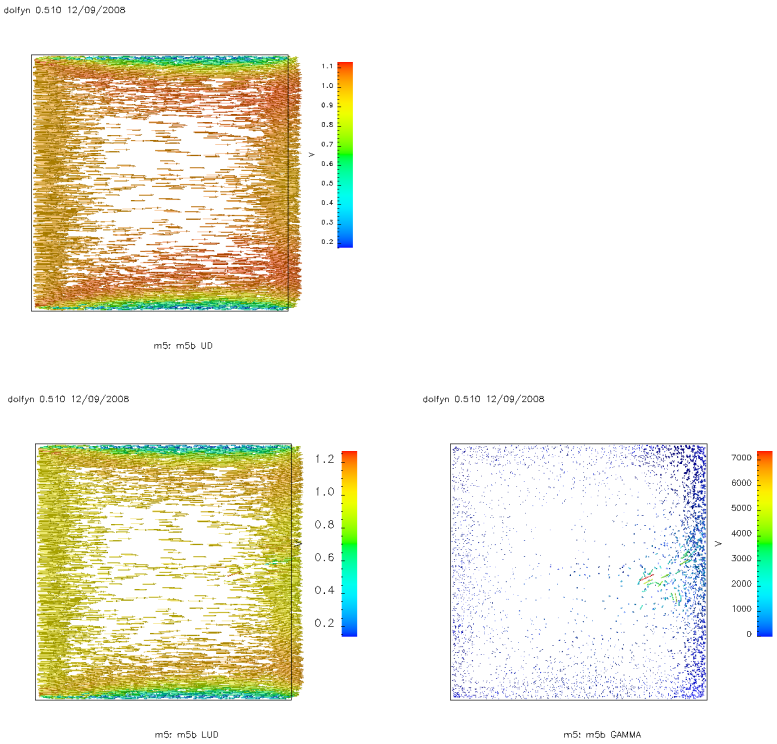


Figure C.10: m5b left to right with UD, CD(0.8), LUD, Gamma

C.9 Stagnation flow with tetrahedral cells only

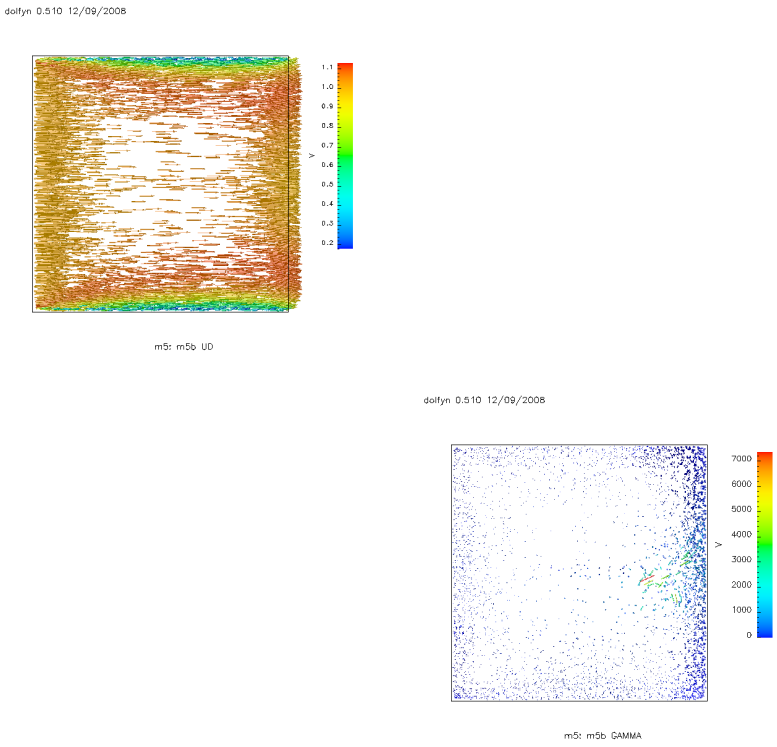


Figure C.11: m5c stagnation flow with UD, CD(0.8), LUD, Gamma

C.10 Stagnation flow with polyhedral cells

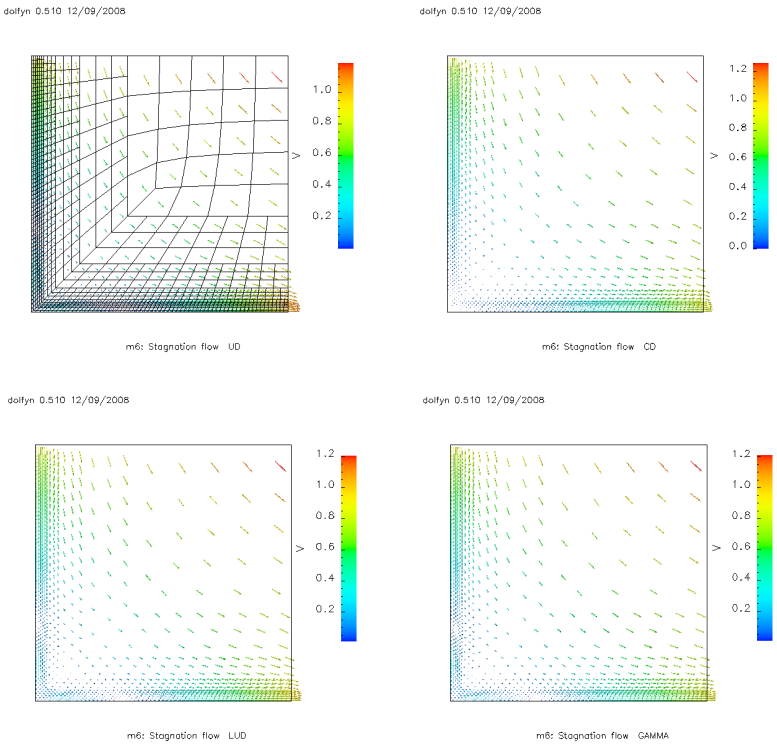


Figure C.12: m6 stagnation flow with UD, CD(0.8), LUD, Gamma

D Mathematical toolbox

D.1 Gauss' theorem

Let V be a volume in $\mathcal{R}^3$ with a smooth and closed surface S with an *o*utwards facing normal $\vec{n} = (n_1, n_2, n_3)$. Further let $\vec{w} = (w_1, w_2, w_3)$ be a differentiable vector field then

$$\iiint_V \text{div}(\vec{w})\, dx\, dy\, dz = \iint_S \vec{w} \cdot \vec{n}\, dS \tag{D.1}$$

The integral $\iint_S \vec{w} \cdot \vec{n}\, dS$ (or $\iint_S \vec{w} \cdot \vec{n}\, d\Omega$, or $\oint_S \vec{w} \cdot \vec{n}\, d\Omega$) is called the *flux* of $\vec{w}$ through S.

D.2 Sum of two vectors

Graphically, a vector is represented by an arrow, defining the direction, and the length of the arrow defines the vector's magnitude. If one end of the arrow is the origin O and the tip of the arrow is A. Then the vector may be represented algebraically by OA (also as $\vec{a}$, $\overline{a}$, or $\mathbf{a}$ (bold)) with magnitude $||\vec{a}||$.

Two vectors, $\vec{a}$ and $\vec{b}$ are equal if they have the same magnitude and direction, regardless of whether they have the same initial points. A vector having the same magnitude as $\vec{a}$ but in the opposite direction to $\vec{a}$ is denoted by $-\vec{a}$.

The sum of two vectors is defined by:

$$\vec{c} = \vec{a} + \vec{b} \tag{D.2}$$

or

$$c_i = a_i + b_i$$

when $\vec{a} + \vec{b} = \vec{0}$ then $\vec{b} = -\vec{a}$ or $b_i = -a_i$. Following properties hold:

1. $\vec{a} + \vec{b} = \vec{b} + \vec{a}$ (commutative property)
2. $\vec{a} + (\vec{b} + \vec{c}) = (\vec{a} + \vec{b}) + \vec{c}$ (associative property)
3. $\vec{a} + \vec{0} = \vec{a}$
4. $\vec{a} + (-\vec{a}) = \vec{0}$

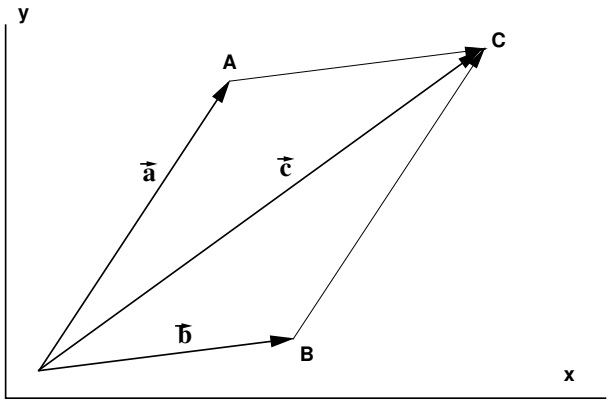


Figure D.1: Sum of two vectors

Vector subtraction is defined as the difference of two vectors, $\vec{a} - \vec{b}$, is a vector $\vec{c}$ that is, $\vec{c} = \vec{a} - \vec{b}$, or $\vec{c} = \vec{a} + (-\vec{b})$.Thus vector subtraction can be represented as a vector addition.

D.3 Multiplying a vector

Is $\lambda \epsilon \mathcal{R}$, $(\lambda \neq 0)$, and $\vec{a}$ a vector, $\vec{a} \neq \vec{0}$, then the length of $||\lambda\vec{a}|| = |\lambda|\ ||\vec{a}||$. For $\lambda = 0$, or $\vec{a} = 0$, is $\lambda\vec{a} = \vec{0}$.

Following multiplication properties hold:

1. $\lambda(\mu\vec{a}) = (\lambda\mu)\vec{a}$ (associative property)
2. $1\vec{a} = \vec{a}$
3. $\lambda(\vec{a} + \vec{b}) = \lambda\vec{a} + \lambda\vec{b}$ (distributive property)
4. $(\lambda + \mu)\vec{a} = \lambda\vec{a} + \mu\vec{a}$ (distributive property)

D.4 Length of a vector and vector products

Is $\lambda \epsilon \mathcal{R}$, $(\lambda \neq 0)$, and $\vec{a}$ a vector, $\vec{a} \neq \vec{0}$, then the length of $||\lambda\vec{a}|| = |\lambda|\ ||\vec{a}||$. For $\lambda = 0$, or $\vec{a} = 0$, is $\lambda\vec{a} = \vec{0}$.

A unit vector is one which has a magnitude of 1. Is $\vec{a}$ a vector $(\vec{a} \neq \vec{0})$, then the unit vector in the direction of $\vec{a}$ is $\hat{a}$. Suppose $\hat{a} = \lambda\vec{a}$ $(\lambda > 0)$, then

$$1 = ||\hat{a}|| = \lambda\vec{a} \tag{D.3}$$

and so

$$\lambda = \frac{1}{||\vec{a}||}$$

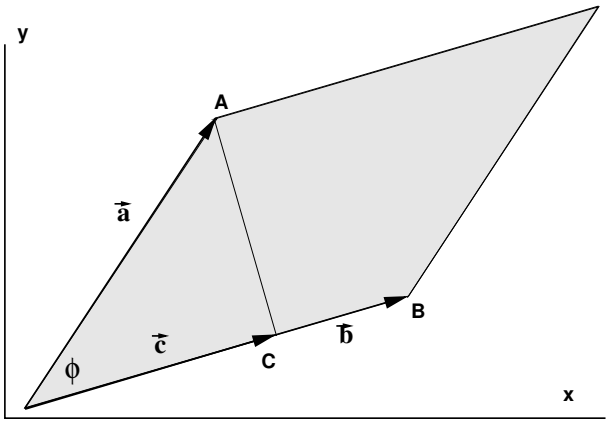


Figure D.2: Inner product of two vectors (hashed area of outer product)

The unit vector in the direction of $\vec{a}$ is

$$\hat{a} = \frac{\vec{a}}{||\vec{a}||} \tag{D.4}$$

Replacing $\vec{a}$ by $\hat{a}$ is called normalising of $\vec{a}$ (not defined for $\vec{a} = \vec{0}$).

The breaking up of a vector into it's component parts is known as resolving a vector. Notice that the representation of $\vec{a}$ by it's components, eg. a_x and a_y is not unique. Depending on the orientation of the coordinate system with respect to the vector in question, it is possible to have more than one set of components.

The inner product, scalar, or dot product is defined by:

$$\vec{a} \cdot \vec{b} = \sum_i a_i b_i \tag{D.5}$$

which results in a scalar.

The scalar product of two vectors, $\vec{a}$ and $\vec{b}$ denoted by $\vec{a} \cdot \vec{b}$, is defined as the product of the magnitudes of the vectors times the cosine of the angle between them, as illustrated in Figure D.2.

$$\vec{a} \cdot \vec{b} = ||\vec{a}||\ ||\vec{b}||\ \cos\phi \tag{D.6}$$

or

$$\cos\phi = \frac{\vec{a} \cdot \vec{b}}{||\vec{a}||\ ||\vec{b}||} \tag{D.7}$$

And in particular when $\hat{a} \cdot \hat{a} = \hat{b} \cdot \hat{b} = 1$, since the angle between a vector and itself is 0 and the cosine of 0 is 1.

Alternatively, when $\hat{a} \cdot \hat{b} = 0$, since the angle between $\vec{a}$ and $\vec{b}$ is 90° and the cosine of 90° is 0.

In general then, if $\hat{a} \cdot \hat{b} = 0$ and neither the magnitude of $\vec{a}$ nor $\vec{b}$ is 0, then $\vec{a}$ and $\vec{b}$ must be perpendicular.

Following properties hold:

1. $\vec{a} \cdot \vec{b} = \vec{b} \cdot \vec{a}$ (commutative property)
2. $\vec{a} \cdot (\vec{b} + \vec{c}) = \vec{a} \cdot \vec{b} + \vec{a} \cdot \vec{c}$ (distributive property)
3. $(\lambda \vec{a}) \cdot \vec{b} = \vec{a} \cdot (\lambda \vec{b}) = \lambda(\vec{a} \cdot \vec{b})$
4. When $\vec{a} \cdot \vec{a} \geq 0$ and $\vec{a} \cdot \vec{a} = 0$ then $\vec{a} = 0$.

Let a vector normal (a vector perpendicular to a plane) be denoted as $\vec{n}$, a position vector of some point in a plane denoted as $\vec{a}$ and a typical point on the plane denoted as $\vec{r}$. Then the vector $\vec{b} = \vec{r} - \vec{a}$ lies within the plane and is perpendicular to $\vec{n}$. Thus the vector equation for a plane is:

$$(\vec{r} - \vec{a}) \cdot \vec{n} = 0 \tag{D.8}$$

The cross or outer product $\vec{a} \times \vec{b}$ of two vectors $\vec{a}$ and $\vec{b}$ in $\mathcal{R}^3$ is defined as:

$$\vec{a} \times \vec{b} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix} \tag{D.9}$$

which is a vector with length

$$||\vec{a} \times \vec{b}|| = ||\vec{a}||\ ||\vec{b}||\ \sin\phi \tag{D.10}$$

where ϕ is the angle between $\vec{a}$ and $\vec{b}$ ($0 \leq \phi \leq \pi$).

Following properties hold:

1. $\vec{a} \times \vec{b} = -\vec{b} \times \vec{a}$
2. $(\vec{a} + \vec{a'}) \times \vec{b} = \vec{a} \times \vec{b} + \vec{a'} \times \vec{b}$
3. $\vec{a} \times (\vec{b} + \vec{b'}) = \vec{a} \times \vec{b} + \vec{a} \times \vec{b'}$
4. $(\lambda \vec{a}) \times \vec{b} = \vec{a} \times (\lambda \vec{b}) = \lambda(\vec{a} \times \vec{b})$

Using the elementary trigonometric definition of the cosine in Figure D.3:

$$\cos\alpha = \frac{||\vec{u}_n||}{||\vec{u}_P||}$$

From the definition of the dot product:

$$\cos\phi = \frac{\vec{a} \cdot \vec{b}}{||\vec{a}||\ ||\vec{b}||}$$

or here with the surface normal $\vec{x}_{n'}$ facing inwards:

$$\cos\phi = \frac{\vec{x}_{n'} \cdot \vec{u}_P}{||\vec{x}_{n'}||\ ||\vec{u}_P||}$$

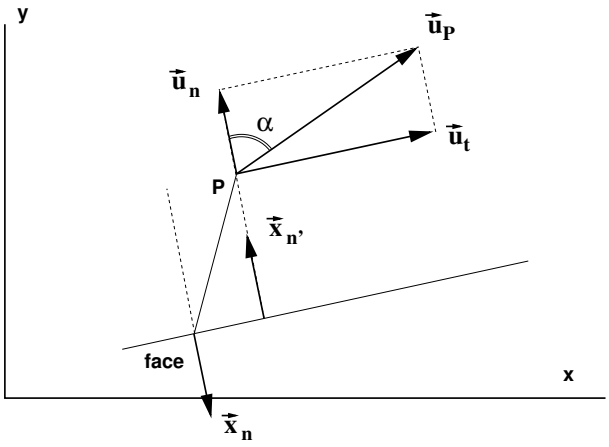


Figure D.3: Vector components near a wall

The length of the normal $\vec{x}_{n'}$ is by definition 1, thus

$$\frac{||\vec{u}_n||}{||\vec{u}_P||} = \frac{\vec{x}_{n'} \cdot \vec{u}_P}{||\vec{u}_P||}$$

or

$$||\vec{u}_n|| = \vec{x}_{n'} \cdot \vec{u}_P \tag{D.11}$$

Now only the length of $\vec{u}_n$ is known ($||\vec{u}_n||$) without a direction. Vector $\vec{u}_n$ points in the direction of $\vec{x}_{n'}$ thus multiplying the normal with the length results in

$$||\vec{u}_n||\ \vec{x}_{n'} = (\vec{x}_{n'} \cdot \vec{u}_P)\ \vec{x}_{n'}$$

or simply

$$\vec{u}_n = (\vec{x}_{n'} \cdot \vec{u}_P)\ \vec{x}_{n'}$$

Next recall that the normal should face outwards, and with $\vec{x}_{n'} = (-\vec{x}_n)$ follows

$$\vec{u}_n = ((-\vec{x}_n) \cdot \vec{u}_P)\ (-\vec{x}_n)$$

Pulling -1 out

$$\vec{u}_n = (\vec{x}_n \cdot \vec{u}_P)(-1)(-1)\vec{x}_n$$

results in

$$\vec{u}_n = (\vec{x}_n \cdot \vec{u}_P)\ \vec{x}_n \tag{D.12}$$

With other words the resulting vector $\vec{u}_n$ is independent of the choice of the normal (pointing inwards or outwards).

The tangential component then follows from an subtraction of the now two known vectors ($\vec{u}_P$ and $\vec{u}_n$):

$$\vec{u}_t = \vec{u}_P - \vec{u}_n \tag{D.13}$$

Bibliography

[1] M.A. Alves, P. Cruz, A. Mendes, , F.D. Magalh aes, and P.J. Pinho, F.T.Oliveira. Adaptive multiresolution approach for solution of hyperbolic PDEs. *Comput. Methods Appl. Mech. Engrg.*, 191:3909–3928, 2002.

[2] M.A. Alves, P.J. Oliveira, and F.T. Pinho. A convergent and universally bounded interpolation scheme for the treatment of advection. *Int. J. Numer. Methods Fluids*, 41:47–75, 2003.

[3] A.A. Amsden. KIVA-3: A KIVA program with block-structured mesh for complex geometries. Technical Report LA-12503-MS, Los Alamos Nat. Lab., March 1993.

[4] A.A. Amsden. KIVA-3V: A block-structured KIVA program for engines with vertical or canted valves. Technical Report LA-13313-MS, Los Alamos Nat. Lab., July 1997.

[5] A.A. Amsden and F.H. Harlow. The SMAC method: A numerical technique for calculating incompressible fluid flows. Technical Report LA-4370, Los Alamos Nat. Lab., February 1970.

[6] A.A. Amsden, P.J. O'Rourke, and T.D. Butler. KIVA-II: A computer program for chemically reactive flows with sprays. Technical Report LA-11560-MS, Los Alamos Nat. Lab., February 1989.

[7] A.A. Amsden, J.D. Ramshaw, L.D. Cloutman, and P.J. O'Rourke. Improvements and extensions to the KIVA computer program. Technical Report LA-10534-MS, Los Alamos Nat. Lab., October 1985.

[8] A.A. Amsden, J.D. Ramshaw, P.J. O'Rourke, and J.K. Dukowicz. KIVA: A computer program for two and three-dimensional fluid flows with chemically reactions and fuel sprays. Technical Report LA-10245-MS, Los Alamos Nat. Lab., February 1985.

[9] J.D. Anderson. *Computational Fluid Dynamics, the basics with applications*. McGraw-Hill, 1995.

[10] M.S. Darwish and F. Moukalled. Normalized variable and space formulation methodology for high-resolution schemes. *Num. Heat Transfer, Part B*, 30:217–237, 1994.

[11] J.H. Ferziger and M. Perîc. *Computational Methods for Fluid Dynamics*. Springer, Berlin, 1996.

[12] J.H. Ferziger and M. Perîc. *Computational methods for fluid dynamics. 3rd rev. ed.* Springer, Berlin, 2002.

[13] J.E. Fromm and F.H. Harlow. Numerical solution of the problem of vortex street development. *Physics of fluids*, 6, July 1963.

[14] R.A. Gentry, B.J. Daly, and A.A. Amsden. KIVA-COAL: A modified version of the KIVA program for calculating the combustion dynamics of a coal-water slurry in a Diesel engine cylinder. Technical Report LA-11045-MS, Los Alamos Nat. Lab., October 1987.

[15] U. Ghia, K. N. Ghia, and C. T. Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of Computational Physics*, 48:387–411, 1982.

[16] A.D Gosman, W.M. Pun, A.K. Runchal, B.D. Spalding, and M. Wolfhstein. *Heat and mass transfer in recirculating flows*. Academic Press, 1969.

[17] F.H. Harlow and C.W. Hirt. Generalized transport theory of anisotropic turbulence. Technical Report LA-4086, Los Alamos Nat. Lab., May 1969.

[18] F.H. Harlow and P.I. Nakayama. Turbulence transport equations. *Physics of fluids*, 10, November 1967.

[19] F.H. Harlow and P.I. Nakayama. Transport of turbulence energy decay rate. Technical Report LA-3854, Los Alamos Nat. Lab., February 1968.

[20] F.H. Harlow and J.E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids*, 8, 1965.

[21] J.O. Hinze. *Turbulence*. McGraw-Hill, 1975.

[22] C.W. Hirt, A.A. Amsden, and J.L. Cook. An arbitrary lagrangian-eulerian method for all flow speeds. *J. Comput. Phys.*, 14:227–253, 1974.

[23] C.W. Hirt, B.D. Nichols, and N.C. Romero. SOLA: A numerical solution algorithm for transient fluid flows. Technical Report LA-5852, Los Alamos Nat. Lab., April 1975.

[24] H. Jasak. *Error analysis and estimation in the Finite Volume method with applications to fluid flows*. PhD thesis, Imperial College, University of London, 1996.

[25] H. Jasak, H.G. Weller, and A.D. Gosman. High resolution NVD differencing scheme for arbitrarily unstructured meshes. *Int. J. Numer. Methods Fluids*, 31(2):431–449, 1999.

[26] N.L. Johnson. The legacy and future of CFD at Los Alamos.

[27] Y. Nakayama. *Visualized flow; fluid mootion in basic and engineering situations revealed by flow visualization*. Pergamon Press, 1988.

[28] B.D. Nichols, C.W. Hirt, and R.S. Hotchkiss. SOLA-VOF: A solution algorithm for transient fluid flow with multiple free boundaries. Technical Report LA-8355, Los Alamos Nat. Lab., August 1980.

[29] S.V. Patankar. *Numerical heat transfer and fluid flow.* Hemisphere Publ. Corp.; McGraw-Hill, 1980.

[30] M. Perîc. *A finite volume method for the prediction of three-dimensional fluid flow in complex ducts*. PhD thesis, Imperial College, University of London, August 1985.

[31] S.B. Pope. *Turbulent flows*. Cambridge University Press, 2000.

[32] C.M. Rhie and W.L. Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal 21 (11)*, 1983.

[33] W.C. Rivard, O.A. Framer, and T.D. Butler. RICE: A computer program for multicomponent chemically reactive flows at all speeds. Technical Report LA-5812, Los Alamos Nat. Lab., March 1975.

[34] P.J. Roache. *Computational fluid dynamics.* Hermosa Publishers, Albuquerque NM, 1972.

[35] P.J. Roache. *Verification and Validation in Computational Science and Engineering.* Hermosa Publishers, Albuquerque NM, 1998.

[36] H. Schlichting. *Boundary-layer theory. Transl. by J. Kestin. 7th ed.* McGraw-Hill, 1979.

[37] C.T. Shaw. *Using computational fluid dynamics.* Prentice Hall, 1992.

[38] B.D. Spalding. A novel finite difference formulation for differential expressions involving both first and second derivates. *Int.J. for numerical methods in engineering*, 4:551–559, 1972.

[39] H. Tennekes and J. L. Lumley. *A First Course in Turbulence*. MIT Press, Cambridge, MA, 1983.

[40] M. Van Dyke. *An Album of Fluid Motion*. The Parabolic Press, 1982.

[41] H.K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics*. Longman, 1995.

[42] H.K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics, The Finite Volume Method*. Pearson, 2nd edition, 2007.

[43] J.E. Welch, F.H. Harlow, J.P. Shannon, and B.J. Daly. The MAC method: A cumputing technique for solving viscous, incompressible, transient fluid-flow problems involving free surfaces. Technical Report LA-3425, Los Alamos Nat. Lab., March 1966.

[44] P. Wesseling. *Principles of computational fluid dynamics.* Springer, Berlin, 2000.

[45] D.C. Wilcox. *Turbulence Modeling for CFD*. DCW Industries, 1993.

[46] M. Zijlema, A. Segal, and P. Wesseling. Finite volume computation of 2D incompressible turbulent flows in general coordinates on staggered grids. Technical Report 94-24, TU Delft, 1994.

Index