



*The Open Source CFD Toolbox*

## Tutorial Guide

Version v2506  
28th June 2025

Copyright © 2025 OpenCFD Limited.

This work is licensed under a **Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License**.

Typeset in L<sup>A</sup>T<sub>E</sub>X.

# License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE (“CCPL” OR “LICENSE”). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

## 1. Definitions

- a. **“Adaptation”** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image (“synching”) will be considered an Adaptation for the purpose of this License.
- b. **“Collection”** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.
- c. **“Distribute”** means to make available to the public the original and copies of the Work through sale or other transfer of ownership.
- d. **“Licensor”** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.
- e. **“Original Author”** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.
- f. **“Work”** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are

assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

- g. **“You”** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
- h. **“Publicly Perform”** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.
- i. **“Reproduce”** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

## 2. Fair Dealing Rights

Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

## 3. License Grant

Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

- a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections; and,
- b. to Distribute and Publicly Perform the Work including as incorporated in Collections.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Adaptations. Subject to 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Section 4(d).

## 4. Restrictions

The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

- 
- a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested.
  - b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
  - c. If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.
  - d. For the avoidance of doubt:
    - i. **Non-waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
    - ii. **Waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,
-

- iii. **Voluntary License Schemes.** The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b).
- e. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation.

## 5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

## 6. Limitation on Liability

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## 7. Termination

- a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
- b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

## 8. Miscellaneous

- a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted

to You under this License.

- b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
- c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
- d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
- e. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

## Trademarks

ANSYS is a registered trademark of ANSYS Inc.

CFX is a registered trademark of Ansys Inc.

CHEMKIN is a registered trademark of Reaction Design Corporation

EnSight is a registered trademark of Computational Engineering International Ltd.

Fluent is a registered trademark of Ansys Inc.

GAMBIT is a registered trademark of Ansys Inc.

Icem-CFD is a registered trademark of Ansys Inc.

I-DEAS is a registered trademark of Structural Dynamics Research Corporation

JAVA is a registered trademark of Sun Microsystems Inc.

Linux is a registered trademark of Linus Torvalds

OpenFOAM is a registered trademark of OpenCFD Ltd

ParaView is a registered trademark of Kitware

STAR-CD is a registered trademark of Computational Dynamics Ltd.

UNIX is a registered trademark of The Open Group



# Contents

<b>Copyright Notice</b>	<b>T-2</b>
<b>Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported Licence</b>	<b>T-3</b>
1. Definitions	T-3
2. Fair Dealing Rights	T-4
3. License Grant	T-4
4. Restrictions	T-4
5. Representations, Warranties and Disclaimer	T-6
6. Limitation on Liability	T-6
7. Termination	T-6
8. Miscellaneous	T-6
<b>Trademarks</b>	<b>T-8</b>
<b>Contents</b>	<b>T-9</b>
<b>1 Introduction</b>	<b>T-13</b>
1.1 Getting started	T-13
1.1.1 Note for Windows Users	T-14
<b>2 Incompressible flow</b>	<b>T-15</b>
2.1 Lid-driven cavity flow	T-16
2.1.1 Pre-processing	T-16
2.1.1.1 Mesh generation	T-16
2.1.1.2 Boundary and initial conditions	T-18
2.1.1.3 Physical properties	T-19
2.1.1.4 Control	T-20
2.1.1.5 Discretisation and linear-solver settings	T-21
2.1.2 Viewing the mesh	T-21
2.1.3 Running an application	T-23
2.1.4 Post-processing	T-23
2.1.4.1 Isosurface and contour plots	T-23
2.1.4.2 Vector plots	T-25
2.1.4.3 Streamline plots	T-25
2.1.5 Increasing the mesh resolution	T-28
2.1.5.1 Creating a new case using an existing case	T-28
2.1.5.2 Creating the finer mesh	T-28
2.1.5.3 Mapping the coarse mesh results onto the fine mesh	T-28
2.1.5.4 Control adjustments	T-29
2.1.5.5 Running the code as a background process	T-29

2.1.5.6	Vector plot with the refined mesh . . . . .	T-30
2.1.5.7	Plotting graphs . . . . .	T-30
2.1.6	Introducing mesh grading . . . . .	T-32
2.1.6.1	Creating the graded mesh . . . . .	T-33
2.1.6.2	Changing time and time step . . . . .	T-34
2.1.6.3	Mapping fields . . . . .	T-35
2.1.7	Increasing the Reynolds number . . . . .	T-35
2.1.7.1	Pre-processing . . . . .	T-35
2.1.7.2	Running the code . . . . .	T-35
2.1.8	High Reynolds number flow . . . . .	T-36
2.1.8.1	Pre-processing . . . . .	T-37
2.1.8.2	Running the code . . . . .	T-38
2.1.9	Changing the case geometry . . . . .	T-39
2.1.10	Post-processing the modified geometry . . . . .	T-41
2.2	Flow around a cylinder . . . . .	T-45
2.2.1	Problem specification . . . . .	T-45
2.2.2	Note on <code>potentialFoam</code> . . . . .	T-46
2.2.3	Mesh generation . . . . .	T-46
2.2.4	Boundary conditions and initial fields . . . . .	T-49
2.2.5	Running the case . . . . .	T-49
2.3	Magnetohydrodynamic flow of a liquid . . . . .	T-53
2.3.1	Problem specification . . . . .	T-53
2.3.2	Mesh generation . . . . .	T-54
2.3.3	Running the case . . . . .	T-55
<b>3</b>	<b>Compressible flow</b>	<b>T-57</b>
3.1	Steady turbulent flow over a backward-facing step . . . . .	T-58
3.1.1	Problem specification . . . . .	T-58
3.1.2	Mesh generation . . . . .	T-59
3.1.3	Boundary conditions and initial fields . . . . .	T-62
3.1.4	Case control . . . . .	T-62
3.1.5	Running the case and post-processing . . . . .	T-62
3.2	Supersonic flow over a forward-facing step . . . . .	T-64
3.2.1	Problem specification . . . . .	T-64
3.2.2	Mesh generation . . . . .	T-65
3.2.3	Running the case . . . . .	T-67
3.2.4	Exercise . . . . .	T-67
3.3	Decompression of a tank internally pressurised with water . . . . .	T-68
3.3.1	Problem specification . . . . .	T-68
3.3.2	Mesh Generation . . . . .	T-69
3.3.3	Preparing the Run . . . . .	T-71
3.3.4	Running the case . . . . .	T-72
3.3.5	Improving the solution by refining the mesh . . . . .	T-72
<b>4</b>	<b>Multiphase flow</b>	<b>T-75</b>
4.1	Breaking of a dam . . . . .	T-76
4.1.1	Problem specification . . . . .	T-76
4.1.2	Mesh generation . . . . .	T-76
4.1.3	Boundary conditions . . . . .	T-78
4.1.4	Setting initial field . . . . .	T-79
4.1.5	Fluid properties . . . . .	T-79

4.1.6	Turbulence modelling . . . . .	T-81
4.1.7	Time step control . . . . .	T-81
4.1.8	Discretisation schemes . . . . .	T-82
4.1.9	Linear-solver control . . . . .	T-82
4.1.10	Running the code . . . . .	T-83
4.1.11	Post-processing . . . . .	T-83
4.1.12	Running in parallel . . . . .	T-83
4.1.13	Post-processing a case run in parallel . . . . .	T-86
<b>5</b>	<b>Stress analysis</b>	<b>T-89</b>
5.1	Stress analysis of a plate with a hole . . . . .	T-90
5.1.1	Problem specification . . . . .	T-90
5.1.2	Mesh generation . . . . .	T-91
5.1.2.1	Boundary and initial conditions . . . . .	T-93
5.1.2.2	Mechanical properties . . . . .	T-95
5.1.2.3	Thermal properties . . . . .	T-95
5.1.2.4	Control . . . . .	T-95
5.1.2.5	Discretisation schemes and linear-solver control . . . . .	T-96
5.1.3	Running the code . . . . .	T-97
5.1.4	Post-processing . . . . .	T-98
5.1.5	Exercises . . . . .	T-99
5.1.5.1	Increasing mesh resolution . . . . .	T-99
5.1.5.2	Introducing mesh grading . . . . .	T-100
5.1.5.3	Changing the plate size . . . . .	T-100
<b>Index</b>		<b>T-101</b>



# Chapter 1

## Introduction

This guide details the process of setup, simulation and post-processing for some OpenFOAM test cases, with the principal aim of introducing a user to the basic procedures of running OpenFOAM. The `$FOAM_TUTORIALS` directory contains many more cases that demonstrate the use of all the solvers and many utilities supplied with OpenFOAM. Before attempting to run the tutorials, the user must first make sure that they have installed OpenFOAM correctly.

The tutorial cases describe the use of the `blockMesh` pre-processing tool, case setup and running OpenFOAM solvers and post-processing using `paraFoam`. Those users with access to third-party post-processing tools supported in OpenFOAM have an option: either they can follow the tutorials using `paraFoam`; or refer to the User Guide for details on post-processing with external applications.

Copies of all tutorials are available from the `tutorials` directory of the OpenFOAM installation. The tutorials are organised into a set of directories according to the type of flow and then subdirectories according to solver. For example, all the `icoFoam` cases are stored within a subdirectory `incompressible/icoFoam`, where *incompressible* indicates the type of flow. If the user wishes to run a range of example cases, it is recommended that the user copy the `tutorials` directory into their local `run` directory. They can be easily copied by typing:

```
mkdir -p $FOAM_RUN
cp -r $FOAM_TUTORIALS $FOAM_RUN
```

### 1.1 Getting started

An OpenFOAM case requires definitions for the mesh, initial fields, physical models, control parameters, etc. As described in the User Guide section [2.1](#), OpenFOAM data is stored in a set of files within a case directory rather than in a single case file. The case directory is given a suitably descriptive name, *e.g.* the first example case for this tutorial guide is simply named `cavity`, under which the required information is located in the three directories:

- *constant*
- *system*, and
- initial time directory, *e.g.* `0`.

Editing files is possible in OpenFOAM because the I/O uses a plain text dictionary format with keywords that convey sufficient meaning to be understood by even the least

experienced users. Many editors are available for both Linux and Windows environments, *e.g.* on Ubuntu the default GUI-based editor is **gedit**, and default terminal editor is **nano**. Other popular text editors include **vim**, **emacs**, **kate**, and **atom**.

### 1.1.1 Note for Windows Users

When using a shared directory, *e.g.* between Windows and Docker, users may prefer to use a Windows-based text editor. However, care should be taken not to change the encoding of the text files to ensure that they remain readable by OpenFOAM. Good choices may be **atom** and **PSPad** — both are free and automatically keep the correct encoding.

# Chapter 2

## Incompressible flow

## 2.1 Lid-driven cavity flow

Tutorial path:

- [\\$FOAM\\_TUTORIALS/incompressible/icoFoam/cavity/cavity](#)

This tutorial will describe how to pre-process, run and post-process a case involving isothermal, incompressible flow in a two-dimensional square domain. The geometry is shown in Figure 2.1 in which all the boundaries of the square are walls. The top wall moves in the  $x$ -direction at a speed of 1 m/s while the other 3 are stationary. Initially, the flow will be assumed laminar and will be solved on a uniform mesh using the **icoFoam** solver for laminar, isothermal, incompressible flow. During the course of the tutorial, the effect of increased mesh resolution and mesh grading towards the walls will be investigated. Finally, the flow Reynolds number will be increased and the **pisoFoam** solver will be used for turbulent, isothermal, incompressible flow.

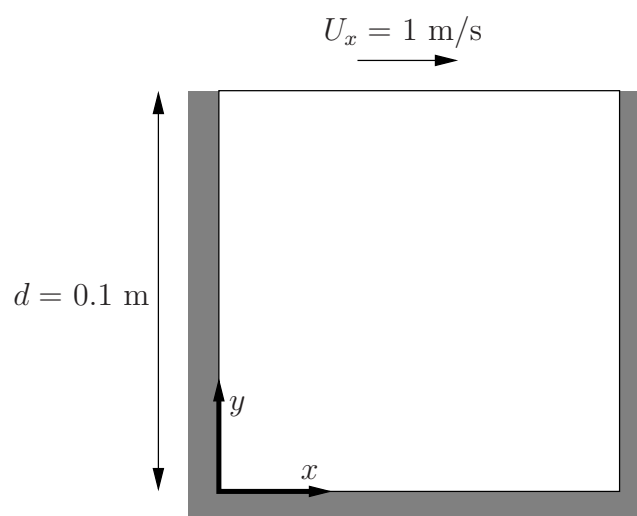


Figure 2.1: Geometry of the lid driven cavity.

### 2.1.1 Pre-processing

In preparation of editing case files and running the first **cavity** case, the user should change to the case directory

```
cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity/cavity
```

#### 2.1.1.1 Mesh generation

OpenFOAM always operates in a 3 dimensional Cartesian coordinate system and all geometries are generated in 3 dimensions. OpenFOAM solves the case in 3 dimensions by default but can be instructed to solve in 2 dimensions by specifying a ‘special’ **empty** boundary condition on boundaries normal to the (3rd) dimension for which no solution is required. Here, the mesh must be 1 cell layer thick, and the **empty** patches planar.

The **cavity** domain consists of a square of side length  $d = 0.1$  m in the  $x$ - $y$  plane. A uniform mesh of 20 by 20 cells will be used initially. The block structure is shown in Figure 2.2. The **blockMesh** mesh generator supplied with OpenFOAM generates meshes from a description specified in an input dictionary, **blockMeshDict** located in the **system** directory for a given case. The **blockMeshDict** entries for this case are as follows:



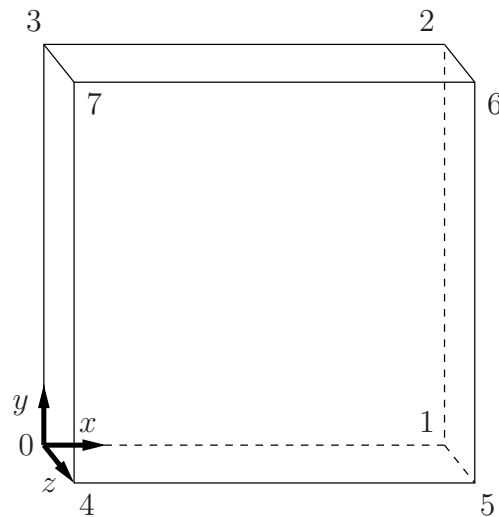


Figure 2.2: Block structure of the mesh for the cavity.

```

1  /*-----*-- C++ *--*/
2  |=====|
3  | \      / | F i e l d      | OpenFOAM: The Open Source CFD Toolbox
4  |  \    /  | O p e r a t i o n | Version: v2506
5  |   \  /   | A n d             | Website: www.openfoam.com
6  |    \/    | M a n i p u l a t i o n |
7  /*-----*--*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        blockMeshDict;
14 }
15 // *****
16
17 scale 0.1;
18
19 vertices
20 (
21     (0 0 0)
22     (1 0 0)
23     (1 1 0)
24     (0 1 0)
25     (0 0 0.1)
26     (1 0 0.1)
27     (1 1 0.1)
28     (0 1 0.1)
29 );
30
31 blocks
32 (
33     hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
34 );
35
36 edges
37 (
38 );
39
40 boundary
41 (
42     movingWall
43     {
44         type wall;
45         faces
46         (
47             (3 7 6 2)
48         );
49     }
50     fixedWalls
51     {
52         type wall;
53         faces
54         (
55             (0 4 7 3)
56             (2 6 5 1)
57             (1 5 4 0)

```

```

58     );
59     }
60     frontAndBack
61     {
62         type empty;
63         faces
64         (
65             (0 3 2 1)
66             (4 5 6 7)
67         );
68     }
69 );
70
71
72 // *****

```

The file first contains header information in the form of a banner (lines 1-7), then file information contained in a *FoamFile* sub-dictionary, delimited by curly braces (`{...}`).

*For the remainder of the manual:*

For the sake of clarity and to save space, file headers, including the banner and *FoamFile* sub-dictionary, will be removed from verbatim quoting of case files

The file first specifies the list of coordinates representing the block **vertices**; These are in arbitrary units, and can be scaled to the real problem dimensions using the **scale** entry, *e.g.*

```
scale          0.1;
```

The next section defines the **blocks** (here, only 1) using the vertex indices, *i.e.* index 0 for vertex 0, index 1 for vertex 1, and so on, and the number of cells within it in each of the 3 co-ordinate directions, and the cell spacing. The final section defines the boundary patches. Please refer to the User Guide section 4.3 to understand the meaning of the entries in the *blockMeshDict* file.

The mesh is generated by running **blockMesh** on this *blockMeshDict* file. From within the case directory, this is done, simply by typing in the terminal:

```
blockMesh
```

The running status of **blockMesh** is reported in the terminal window. Any mistakes in the *blockMeshDict* file are picked up by **blockMesh** and the resulting error message directs the user to the line in the file where the problem occurred. There should be no error messages at this stage.

### 2.1.1.2 Boundary and initial conditions

Once the mesh generation is complete, the user can look at this initial fields set up for this case. The case is set up to start at time  $t = 0$  s, so the initial field data is stored in a *0* sub-directory of the *cavity* directory. The *0* sub-directory contains 2 files, *p* and *U*, one for each of the pressure (*p*) and velocity (**U**) fields whose initial values and boundary conditions must be set. Let us examine file *p*:

```

17 dimensions      [0 2 -2 0 0 0 0];
18
19 internalField    uniform 0;
20
21 boundaryField
22 {
23     movingWall
24     {
25         type      zeroGradient;
26     }

```

```

27
28     fixedWalls
29     {
30         type            zeroGradient;
31     }
32
33     frontAndBack
34     {
35         type            empty;
36     }
37 }
38
39
40 // ***** //

```

There are 3 principal entries in field data files:

**dimensions** specifies the dimensions of the field, here *kinematic* pressure, *i.e.*  $\text{m}^2 \text{s}^{-2}$  (see User Guide section 2.2.6 for more information);

**internalField** the internal field data which can be **uniform**, described by a single value; or **nonuniform**, where all the values of the field must be specified (see User Guide section 2.2.8 for more information);

**boundaryField** the boundary field data that includes boundary conditions and data for all the boundary patches (see User Guide section 2.2.8 for more information).

For this case **cavity**, the boundary consists of walls only, split into 2 patches named: (1) **fixedWalls** for the fixed sides and base of the cavity; (2) **movingWall** for the moving top of the cavity. As walls, both are given a **zeroGradient** boundary condition for **p**, meaning “the normal gradient of pressure is zero”. The **frontAndBack** patch represents the front and back planes of the 2D case and therefore must be set as **empty**.

In this case, as in most we encounter, the initial fields are set to be uniform. Here the pressure is kinematic, and as an incompressible case, its absolute value is not relevant, so is set to **uniform 0** for convenience.

The user can similarly examine the velocity field in the *0/U* file. The **dimensions** are those expected for velocity, the internal field is initialised as uniform zero, which in the case of velocity must be expressed by 3 vector components, *i.e.* **uniform (0 0 0)** (see User Guide section 2.2.5 for more information).

The boundary field for velocity requires the same boundary condition for the **frontAndBack** patch. The other patches are walls: a no-slip condition is assumed on the **fixedWalls**, hence a **fixedValue** condition with a **value** of **uniform (0 0 0)**. The top surface moves at a speed of 1 m/s in the *x*-direction so requires a **fixedValue** condition also but with **uniform (1 0 0)**.

### 2.1.1.3 Physical properties

The physical properties for the case are stored in dictionaries whose names are given the suffix *...Properties*, located in the **Dictionaries** directory tree. For an **icoFoam** case, the only property that must be specified is the kinematic viscosity which is stored from the **transportProperties** dictionary. The user can check that the kinematic viscosity is set correctly by opening the **transportProperties** dictionary to view/edit its entries. The keyword for kinematic viscosity is **nu**, the phonetic label for the Greek symbol  $\nu$  by which it is represented in equations. Initially this case will be run with a Reynolds number of 10, where the Reynolds number is defined as:

$$Re = \frac{d|\mathbf{U}|}{\nu} \quad (2.1)$$

where  $d$  and  $|\mathbf{U}|$  are the characteristic length and velocity respectively and  $\nu$  is the kinematic viscosity. Here  $d = 0.1$  m,  $|\mathbf{U}| = 1$  m s<sup>-1</sup>, so that for  $Re = 10$ ,  $\nu = 0.01$  m<sup>2</sup> s<sup>-1</sup>. The correct file entry for kinematic viscosity is thus specified below:

```

17  nu                0.01;
18
19
20  // ***** //
```

#### 2.1.1.4 Control

Input data relating to the control of time and reading and writing of the solution data are read in from the *controlDict* dictionary. The user should view this file; as a case control file, it is located in the *system* directory.

The start/stop times and the time step for the run must be set. OpenFOAM offers great flexibility with time control which is described in full in the User Guide section 6.1. In this tutorial we wish to start the run at time  $t = 0$  which means that OpenFOAM needs to read field data from a directory named *0* — see User Guide section 2.1 for more information of the case file structure. Therefore we set the **startFrom** keyword to **startTime** and then specify the **startTime** keyword to be 0.

For the end time, we wish to reach the steady state solution where the flow is circulating around the cavity. As a general rule, the fluid should pass through the domain 10 times to reach steady state in laminar flow. In this case the flow does not pass through this domain as there is no inlet or outlet, so instead the end time can be set to the time taken for the lid to travel ten times across the cavity, *i.e.* 1 s; in fact, with hindsight, we discover that 0.5 s is sufficient so we shall adopt this value. To specify this end time, we must specify the **stopAt** keyword as **endTime** and then set the **endTime** keyword to 0.5.

Now we need to set the time step, represented by the keyword **deltaT**. To achieve temporal accuracy and numerical stability when running *icoFoam*, a Courant number of less than 1 is required. The Courant number is defined for one cell as:

$$Co = \frac{\delta t |\mathbf{U}|}{\delta x} \quad (2.2)$$

where  $\delta t$  is the time step,  $|\mathbf{U}|$  is the magnitude of the velocity through that cell and  $\delta x$  is the cell size in the direction of the velocity. The flow velocity varies across the domain and we must ensure  $Co < 1$  everywhere. We therefore choose  $\delta t$  based on the worst case: the *maximum*  $Co$  corresponding to the combined effect of a large flow velocity and small cell size. Here, the cell size is fixed across the domain so the maximum  $Co$  will occur next to the lid where the velocity approaches 1 m s<sup>-1</sup>. The cell size is:

$$\delta x = \frac{d}{n} = \frac{0.1}{20} = 0.005 \text{ m} \quad (2.3)$$

Therefore to achieve a Courant number less than or equal to 1 throughout the domain the time step **deltaT** must be set to less than or equal to:

$$\delta t = \frac{Co \delta x}{|\mathbf{U}|} = \frac{1 \times 0.005}{1} = 0.005 \text{ s} \quad (2.4)$$

As the simulation progresses we wish to write results at certain intervals of time that we can later view with a post-processing package. The **writeControl** keyword presents several options for setting the time at which the results are written; here we select the **timeStep** option which specifies that results are written every  $n$ th time step where the value  $n$  is specified under the **writeInterval** keyword. Let us decide that we wish to

write our results at times 0.1, 0.2, ..., 0.5 s. With a time step of 0.005 s, we therefore need to output results at every 20th time step and so we set `writeInterval` to 20.

OpenFOAM creates a new directory *named after the current time*, e.g. 0.1 s, on each occasion that it writes a set of data, as discussed in full in User Guide section 2.1. In the `icoFoam` solver, it writes out the results for each field,  $U$  and  $p$ , into the time directories. For this case, the entries in the *controlDict* are shown below:

```

17  application      icoFoam;
18
19  startFrom        startTime;
20
21  startTime        0;
22
23  stopAt           endTime;
24
25  endTime          0.5;
26
27  deltaT           0.005;
28
29  writeControl      timeStep;
30
31  writeInterval     20;
32
33  purgeWrite       0;
34
35  writeFormat       ascii;
36
37  writePrecision    6;
38
39  writeCompression off;
40
41  timeFormat        general;
42
43  timePrecision     6;
44
45  runtimeModifiable true;
46
47
48  // ***** //
```

### 2.1.1.5 Discretisation and linear-solver settings

The user specifies the choice of finite volume discretisation schemes in the *fvSchemes* dictionary in the *system* directory. The specification of the linear equation solvers and tolerances and other algorithm controls is made in the *fvSolution* dictionary, similarly in the *system* directory. The user is free to view these dictionaries but we do not need to discuss all their entries at this stage except for `pRefCell` and `pRefValue` in the *PISO* sub-dictionary of the *fvSolution* dictionary. In a closed incompressible system such as the cavity, pressure is relative: it is the pressure range that matters not the absolute values. In cases such as this, the solver sets a reference level by `pRefValue` in cell `pRefCell`. In this example both are set to 0. Changing either of these values will change the absolute pressure field, but not, of course, the relative pressures or velocity field.

### 2.1.2 Viewing the mesh

Before the case is run it is a good idea to view the mesh to check for any errors. The mesh is viewed in `paraFoam`, the post-processing tool supplied with OpenFOAM. The `paraFoam` post-processing is started by typing in the terminal from within the case directory

```
paraFoam
```

Alternatively, it can be launched from another directory location with an optional `-case` argument giving the case directory, e.g.

```
paraFoam -case $FOAM_RUN/tutorials/incompressible/icoFoam/cavity/cavity
```

This launches the ParaView window as shown in Figure ???. In the Pipeline Browser, the user can see that ParaView has opened `cavity.OpenFOAM`, the module for the `cavity` case. **Before clicking the Apply button**, the user needs to select some geometry from the Mesh Parts panel. Because the case is small, it is easiest to select all the data by checking the box adjacent to the Mesh Parts panel title, which automatically checks all individual components within the respective panel. The user should then click the **Apply** button to load the geometry into ParaView. Some general settings are applied as described in User Guide section 7.1.7.1. **Please consult this section about these settings.**

The user should then open the Properties panel that controls the visual representation of the selected module. Within the Display panel the user should do the following as shown in Figure 2.3: (1) set Color By Solid Color; (2) click Edit and select an appropriate colour *e.g.* black (for a white background); (3) select Wireframe from the Representation menu. The background colour can be set by selecting `Settings...` from Edit in the top menu panel.

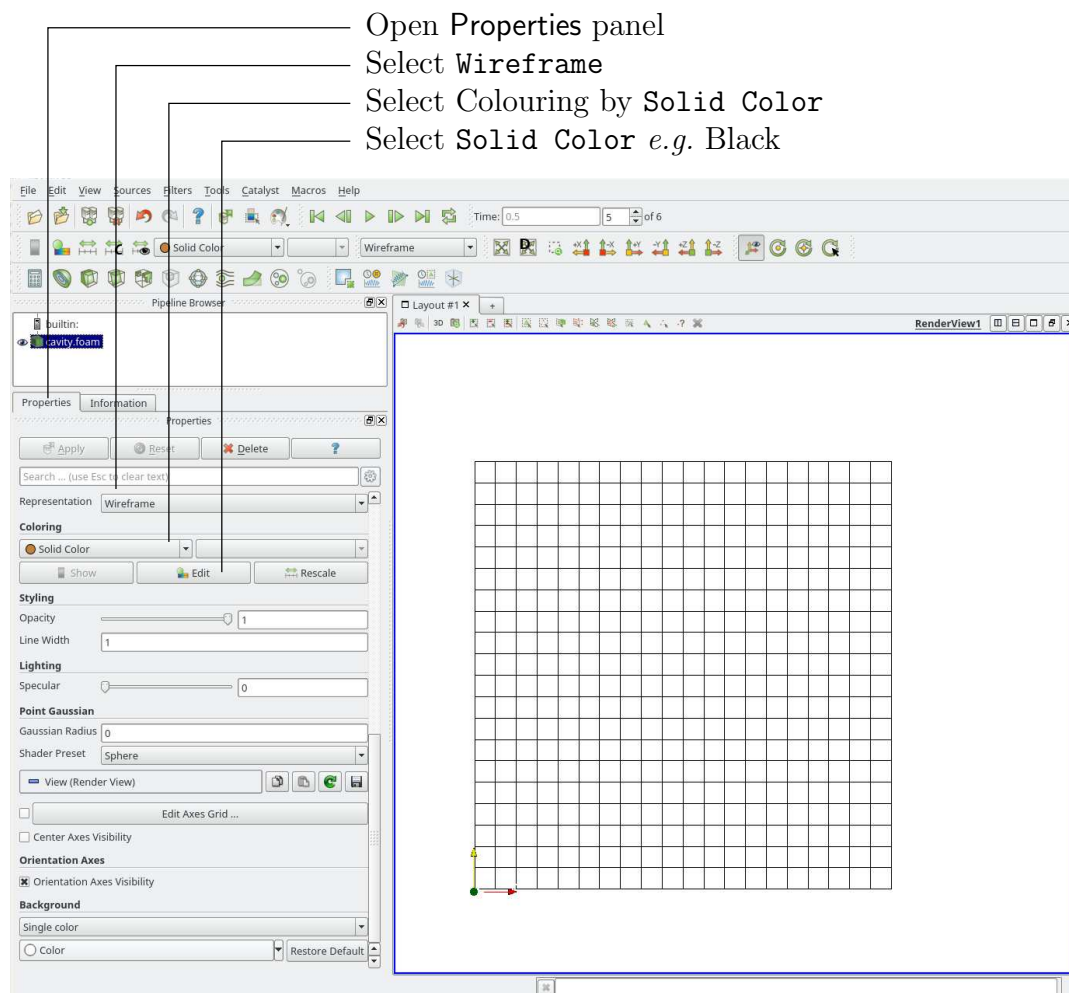


Figure 2.3: Viewing the mesh in paraFoam.

Especially the first time the user starts ParaView, **it is recommended** that they manipulate the view as described in User Guide section 7.1.7. In particular, since this is a 2D case, it is recommended that **Camera Parallel Projection** is selected. To do so, click on the **Toggle Advanced Properties** to show the option towards the bottom on the panel. The **Orientation Axes** can be toggled on and off in the **Annotation** window or moved by drag and drop with the mouse.

### 2.1.3 Running an application

Like any UNIX/Linux executable, OpenFOAM applications can be run in two ways: as a foreground process, *i.e.* one in which the shell waits until the command has finished before giving a command prompt; as a background process, one which does not have to be completed before the shell accepts additional commands.

On this occasion, we will run `icoFoam` in the foreground. The `icoFoam` solver is executed either by entering the case directory and typing

```
icoFoam
```

at the command prompt, or with the optional `-case` argument giving the case directory, *e.g.*

```
icoFoam -case $FOAM_RUN/tutorials/incompressible/icoFoam/cavity/cavity
```


The progress of the job is written to the terminal window. It tells the user the current time, maximum Courant number, initial and final residuals for all fields.



### 2.1.4 Post-processing

As soon as results are written to time directories, they can be viewed using `paraFoam`. Return to the `paraFoam` window and select the **Properties** panel for the `cavity.OpenFOAM` case module. If the correct window panels for the case module do not seem to be present at any time, please ensure that: `cavity.OpenFOAM` is highlighted in blue; **eye** button alongside it is switched on to show the graphics are enabled;

To prepare `paraFoam` to display the data of interest, we must first load the data at the required run time of 0.5 s. If the case was run while `ParaView` was open, the output data in time directories will not be automatically loaded within `ParaView`. To load the data the user should click **Refresh Times** in the **Properties** window. The time data will be loaded into `ParaView`.

#### 2.1.4.1 Isosurface and contour plots

To view pressure, the user should open the **Properties** panel since it controls the visual representation of the selected module. To make a simple plot of pressure, the user should select the following, as described in detail in Figure 2.4: select **Surface** from the **Representation** menu; in the **Coloring** panel, select **Color** by  and **Rescale to Data Range**. Now in order to view the solution at  $t = 0.5$  s, the user can use the **VCR Controls** or **Current Time Controls** to change the current time to 0.5. These are located in the toolbars below the menus at the top of the `ParaView` window, as shown in Figure ???. The pressure field solution has, as expected, a region of low pressure at the top left of the cavity and one of high pressure at the top right of the cavity as shown in Figure 2.5.

With the point icon () the pressure field is interpolated across each cell to give a continuous appearance. Instead if the user selects the cell icon, , from the **Coloring** by menu, a single value for pressure will be attributed to each cell so that each cell will be denoted by a single colour with no grading.

A colour bar can be included by either by clicking the **Toggle Color Legend Visibility** button in the **Active Variable Controls** toolbar, or by selecting **Show Color Legend** from the **View** menu. Clicking the **Edit Color Map** button, either in the **Active Variable Controls** toolbar or in the **Color** panel of the **Display** window, the user can set a range



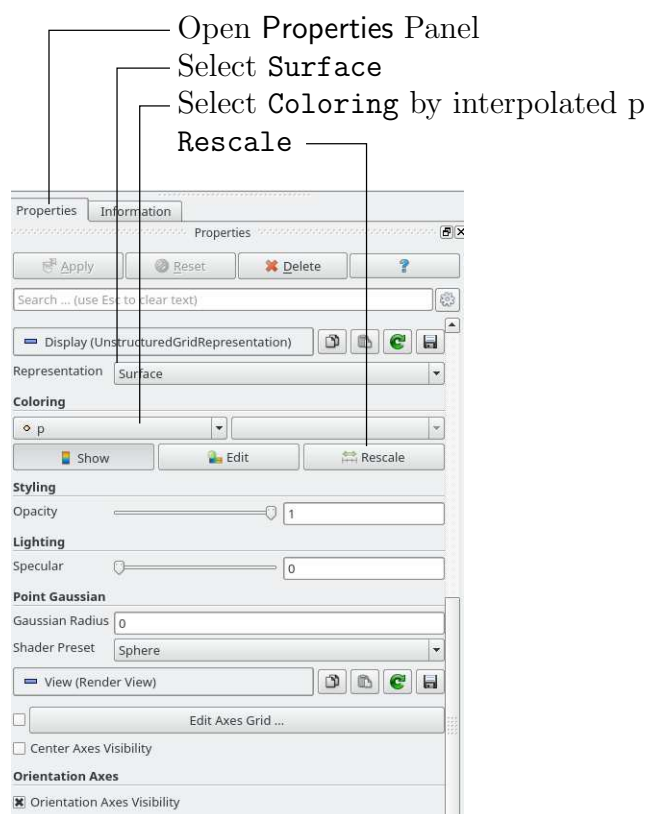


Figure 2.4: Displaying pressure contours for the cavity case.

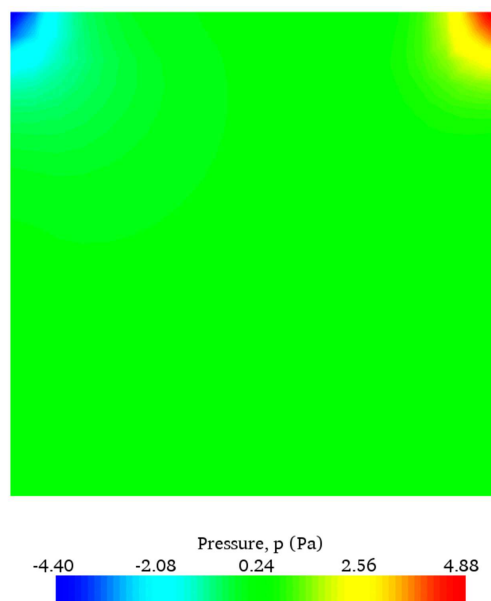


Figure 2.5: Pressures in the cavity case.



of attributes of the colour bar, such as text size, font selection and numbering format for the scale. The colour bar can be located in the image window by drag and drop with the mouse.

New versions of ParaView default to using a colour scale of blue to white to red rather than the more common blue to green to red (rainbow). Therefore *the first time* that the user executes ParaView, they may wish to change the colour scale. This can be done by selecting **Choose Preset** in the **Color Scale Editor** and selecting **Blue to Red Rainbow**. After clicking the **OK** confirmation button, the user can click the **Make Default** button so that ParaView will always adopt this type of colour bar.

If the user rotates the image, they can see that they have now coloured the complete geometry surface by the pressure. In order to produce a genuine contour plot the user should first create a cutting plane, or ‘slice’, through the geometry using the **Slice** filter as described in User Guide section 7.1.8.1. The cutting plane should be centred at (0.05, 0.05, 0.005) and its normal should be set to (0, 0, 1) (click the **Z Normal** button). Having generated the cutting plane, the contours can be created using by the **Contour** filter described in User Guide section 7.1.8.

### 2.1.4.2 Vector plots

Before we start to plot the vectors of the flow velocity, it may be useful to remove other modules that have been created, *e.g.* using the **Slice** and **Contour** filters described above. These can: either be deleted entirely, by highlighting the relevant module in the **Pipeline Browser** and clicking **Delete** in their respective **Properties** panel; or, be disabled by toggling the **eye** button for the relevant module in the **Pipeline Browser**.

We now wish to generate a vector glyph for velocity at the centre of each cell. We first need to filter the data to cell centres as described in User Guide section 7.1.9.1. With the **cavity.OpenFOAM** module highlighted in the **Pipeline Browser**, the user should select **Cell Centers** from the **Filter->Alphabetical** menu and then click **Apply**.

With these **Centers** highlighted in the **Pipeline Browser**, the user should then select **Glyph** from the **Filter->Alphabetical** menu. The **Properties** window panel should appear as shown in Figure 2.6. In the resulting **Properties** panel, the velocity field, **U**, is automatically selected in the **vectors** menu, since it is the only vector field present. By default the **Scale Mode** for the glyphs will be **Vector Magnitude** of velocity but, since we may wish to view the velocities throughout the domain, the user should instead select **off** and **Set Scale Factor** to 0.005. On clicking **Apply**, the glyphs appear coloured by pressure. The user should also select **Show Color Legend** in **Edit Color Map**. The output is shown in Figure 2.7, in which uppercase Times Roman fonts are selected for the **Color Legend** headings and the labels are specified to 2 fixed significant figures by deselecting **Automatic Label Format** and entering **%-#6.2f** in the **Label Format** text box. The background colour is set to white in the **Properties** as described in User Guide section 7.1.7.1.

Note that at the left and right walls, glyphs appear to indicate flow through the walls. On closer examination, however, the user can see that while the flow direction is normal to the wall, its magnitude is 0. This slightly confusing situation is caused by ParaView choosing to orientate the glyphs in the *x*-direction when the glyph scaling **off** and the velocity magnitude is 0.

### 2.1.4.3 Streamline plots

Again, before the user continues to post-process in ParaView, they should disable modules such as those for the vector plot described above. We now wish to plot streamlines of velocity as described in User Guide section 7.1.10.

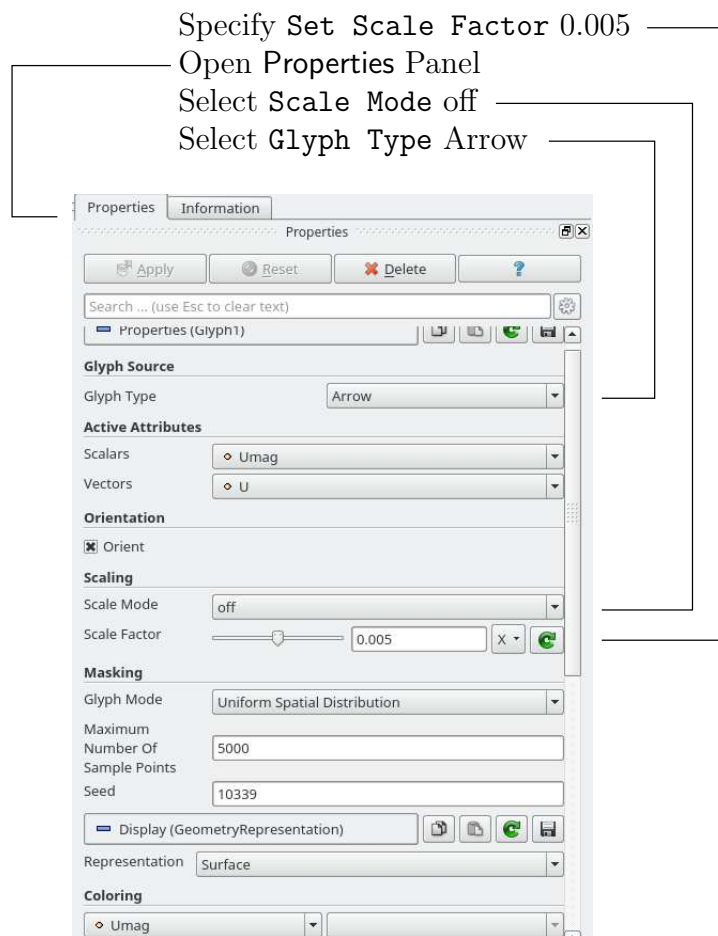


Figure 2.6: Properties panel for the Glyph filter.

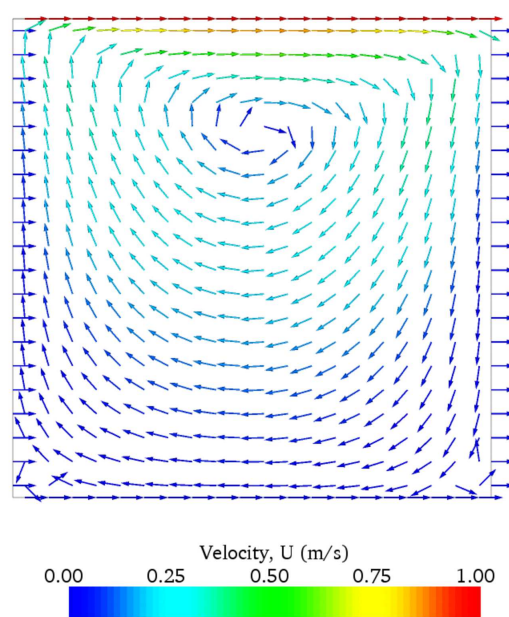


Figure 2.7: Velocities in the cavity case.

- Open the Properties Panel
- Click Toggle Advanced Properties
- Set Integration Step Unit to Cell Length
- Set Initial Step Length to 0.01
- Set Maximum Steps to 1500
- Set Maximum Streamline Length to 0.5
- Set Seed type to High Resolution Line Source
- Set Points and Resolution

The screenshot shows the 'Properties' panel for streamlines in OpenFOAM. The 'Integration Parameters' section includes 'Integration Direction' (BOTH), 'Integrator Type' (Runge-Kutta 2), and 'Integration Step Unit' (Cell Length). The 'Streamline Parameters' section includes 'Maximum Steps' (1500), 'Maximum Streamline Length' (0.5), 'Terminal Speed' (1e-12), and 'Maximum Error' (1e-06). The 'Seeds' section includes 'Seed Type' (High Resolution Line Source), 'Show Line' (unchecked), 'Pick Both Points' (checked), and 'Snap On Mesh Point' (unchecked). The 'Resolution' is set to 21.

**Properties** Information

Apply Reset Delete ?

Search ... (use Esc to clear text)

**Integration Parameters**

Integration Direction BOTH

Integrator Type Runge-Kutta 2

Integration Step Unit Cell Length

Initial Step Length 0.01

Minimum Step Length 0.01

Maximum Step Length 0.01

**Streamline Parameters**

Maximum Steps 1500

Maximum Streamline Length 0.5

Terminal Speed 1e-12

Maximum Error 1e-06

☒ Compute Vorticity

**Seeds**

Seed Type High Resolution Line Source

☐ Show Line

Pick Both Points ☒ Snap On Mesh Point

Point1 0.05 0 0.005

Point2 0.05 0.1 0.005

X Axis

Y Axis

Z Axis

Resolution 21

With the `cavity.OpenFOAM` module highlighted in the Pipeline Browser, the user should then select **Stream Tracer** from the **Filter** menu and then click **Apply**. The **Properties** window panel should appear as shown in Figure 2.8. The **Seed** points should be specified along a **Line Source** running vertically through the centre of the geometry, *i.e.* from (0.05, 0, 0.005) to (0.05, 0.1, 0.005). For the image in this guide we used: a point **Resolution** of 21; **Max Propagation by Length** 0.5; **Initial Step Length by Cell Length** 0.01; and, **Integration Direction** BOTH. The **Runge-Kutta 2 IntegratorType** was used with default parameters.

On clicking **Apply** the tracer is generated. The user should then select **Tube** from the **Filter** menu to produce high quality streamline images. For the image in this report, we used: **Num. sides** 6; **Radius** 0.0003; and, **Radius factor** 10. The streamtubes are coloured by velocity magnitude. On clicking **Apply** the image in Figure 2.9 should be produced.

## 2.1.5 Increasing the mesh resolution

The mesh resolution will now be increased by a factor of two in each direction. The results from the coarser mesh will be mapped onto the finer mesh to use as initial conditions for the problem. The solution from the finer mesh will then be compared with those from the coarser mesh.

### 2.1.5.1 Creating a new case using an existing case

We now wish to create a new case named `cavityFine` that is created from `cavity`. The user should therefore clone the `cavity` case and edit the necessary files. First the user should create a new case directory at the same directory level as the `cavity` case, *e.g.*

```
cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity
mkdir cavityFine
```

The user should then copy the base directories from the `cavity` case into `cavityFine`, and then enter the `cavityFine` case.

```
cp -r cavity/constant cavityFine
cp -r cavity/system cavityFine
cd cavityFine
```

### 2.1.5.2 Creating the finer mesh

We now wish to increase the number of cells in the mesh by using `blockMesh`. The user should open the `blockMeshDict` file in an editor and edit the block specification. The blocks are specified in a list under the `blocks` keyword. The syntax of the block definitions is described fully in User Guide section 4.3.1.3; at this stage it is sufficient to know that following `hex` is first the list of vertices in the block, then a list (or vector) of numbers of cells in each direction. This was originally set to (20 20 1) for the `cavity` case. The user should now change this to (40 40 1) and save the file. The new refined mesh should then be created by running `blockMesh` as before.

### 2.1.5.3 Mapping the coarse mesh results onto the fine mesh

The `mapFields` utility maps one or more fields relating to a given geometry onto the corresponding fields for another geometry. In our example, the fields are deemed ‘consistent’

because the geometry and the boundary types, or conditions, of both source and target fields are identical. We use the `-consistent` command line option when executing `mapFields` in this example.

The field data that `mapFields` maps is read from the time directory specified by `startFrom/startTime` in the *controlDict* of the target case, *i.e.* those **into which** the results are being mapped. In this example, we wish to map the final results of the coarser mesh from case `cavity` onto the finer mesh of case `cavityFine`. Therefore, since these results are stored in the `0.5` directory of `cavity`, the `startTime` should be set to 0.5 s in the *controlDict* dictionary and `startFrom` should be set to `startTime`.

The case is ready to run `mapFields`. Typing `mapFields -help` quickly shows that `mapFields` requires the source case directory as an argument. We are using the `-consistent` option, so the utility is executed from within the *cavityFine* directory by

```
mapFields ../cavity -consistent
```

The utility should run with output to the terminal including:

```
Source: "." "cavity"
Target: "." "cavityFine"

Create databases as time

Source time: 0.5
Target time: 0.5
Create meshes

Source mesh size: 400   Target mesh size: 1600

Consistently creating and mapping fields for time 0.5

    interpolating p
    interpolating U

End
```

#### 2.1.5.4 Control adjustments

To maintain a Courant number of less than 1, as discussed in section 2.1.1.4, the time step must now be halved since the size of all cells has halved. Therefore `deltaT` should be set to 0.0025 s in the *controlDict* dictionary. Field data is currently written out at an interval of a fixed number of time steps. Here we demonstrate how to specify data output at fixed intervals of time. Under the `writeControl` keyword in *controlDict*, instead of requesting output by a fixed number of time steps with the `timeStep` entry, a fixed amount of run time can be specified between the writing of results using the `runTime` entry. In this case the user should specify output every 0.1 and therefore should set `writeInterval` to 0.1 and `writeControl` to `runTime`. Finally, since the case is starting with a the solution obtained on the coarse mesh we only need to run it for a short period to achieve reasonable convergence to steady-state. Therefore the `endTime` should be set to 0.7 s. Make sure these settings are correct and then save the file.

#### 2.1.5.5 Running the code as a background process

The user should experience running `icoFoam` as a background process, redirecting the terminal output to a *log* file that can be viewed later. From the *cavityFine* directory, the user should execute:

```
icoFoam > log &
cat log
```

### 2.1.5.6 Vector plot with the refined mesh

The user can open multiple cases simultaneously in **ParaView**; essentially because each new case is simply another module that appears in the **Pipeline Browser**. There is one minor inconvenience when opening a new case in **ParaView** because there is a prerequisite that the selected data is a file with a name that has an extension. However, in OpenFOAM, each case is stored in a multitude of files with no extensions within a specific directory structure. The solution, that the **paraFoam** script performs automatically, is to create a dummy file with the extension *.OpenFOAM* — hence, the **cavity** case module is called **cavity.OpenFOAM**.

However, if the user wishes to open another case directly from within **ParaView**, they need to create such a dummy file. For example, to load the **cavityFine** case the file would be created by typing at the command prompt:

```
cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity
touch cavityFine/cavityFine.OpenFOAM
```

Now the **cavityFine** case can be loaded into **ParaView** by selecting **Open** from the **File** menu, and having navigated the directory tree, selecting **cavityFine.OpenFOAM**. The user can now make a vector plot of the results from the refined mesh in **ParaView**. The plot can be compared with the **cavity** case by enabling glyph images for both case simultaneously.

### 2.1.5.7 Plotting graphs

The user may wish to visualise the results by extracting some scalar measure of velocity and plotting 2-dimensional graphs along lines through the domain. OpenFOAM is well equipped for this kind of data manipulation. There are numerous utilities that perform specialised data manipulations, and many can be accessed via the **postProcess** utility:

```
postProcess -list
```

returns the list:

```
CourantNo
Lambda2
MachNo
PecletNo
Q
R
XiReactionRate
add
boundaryCloud
cellMax
cellMin
components
div
enstrophy
faceMax
faceMin
flowRatePatch
flowType
```

```

forceCoeffsCompressible
forceCoeffsIncompressible
forcesCompressible
forcesIncompressible
grad
internalCloud
mag
magSqr
minMaxComponents
minMaxMagnitude
patchAverage
patchIntegrate
pressureDifferencePatch
pressureDifferenceSurface
probes
randomise
residuals
scalarTransport
singleGraph
staticPressure
streamFunction
streamlines
subtract
surfaces
totalPressureCompressible
totalPressureIncompressible
turbulenceFields
volFlowRateSurface
vorticity
wallHeatFlux
wallShearStress
writeCellCentres
writeCellVolumes
writeObjects
yPlus

```

The `components` and `mag` functions provide useful scalar measures of velocity.

When the `components` function is run on a case, say *cavity*, it reads the velocity vector field from each time directory and, in the corresponding time directories, writes scalar fields `Ux`, `Uy` and `Uz` representing the  $x$ ,  $y$  and  $z$  components of velocity. Similarly the `mag` function writes a scalar field `magU` to each time directory representing the magnitude of velocity.

The user can run both functions simultaneously, *e.g.* for the cavity case the user should go into the *cavity* directory and execute `postProcess` as follows:

```

cd $FOAM RUN/tutorials/incompressible/icoFoam/cavity/cavity
postProcess -funcs '(components(U) mag(U))'

```

The individual components can be plotted as a graph in `ParaView`. It is quick, convenient and has reasonably good control over labelling and formatting, so the printed output is a fairly good standard. However, to produce graphs for publication, users may



prefer to write raw data and plot it with a dedicated graphing tool, such as **gnuplot** or **Grace/xmgr**. To do this, see section 5.1.4.

*Before commencing plotting*, the user needs to load the newly generated **Ux**, **Uy** and **Uz** fields into **ParaView**. To do this, the user should click the **Refresh** button at the top of the **Properties** panel for the **cavity.OpenFOAM** module which will cause the new fields to be loaded into **ParaView** and appear in the **Volume Fields** window. Ensure the new fields are selected and the changes are applied, *i.e.* click **Apply** again if necessary. *Also*, data is interpolated incorrectly at boundaries if the boundary regions are selected in the **Mesh Parts** panel. Therefore the user should *deselect the patches* in the **Mesh Parts** panel, *i.e.* **movingWall**, **fixedWall** and **frontAndBack**, and apply the changes.

Now, in order to display a graph in **ParaView** the user should select the module of interest, *e.g.* **cavity.OpenFOAM** and apply the **Plot Over Line** filter from the **Filter->Data Analysis** menu. This opens up a new **XY Plot** window below or beside the existing **3D View** window. A **PlotOverLine** module is created in which the user can specify the end points of the line in the **Properties** panel. In this example, the user should position the line vertically up the centre of the domain, *i.e.* from (0.05, 0, 0.005) to (0.05, 0.1, 0.005), in the **Point1** and **Point2** text boxes. The **Resolution** can be set to 100.

On clicking **Apply**, a graph is generated in the **XY Plot** window. In the **Display** panel, the **Attribute Mode** is set to **Point Data** by default. The **Use Data Array** option can be selected for the **X Axis Data**, taking the **arc\_length** option so that the x-axis of the graph represents distance from the base of the cavity.

The user can choose the fields to be displayed in the **Line Series** panel of the **Display** window. From the list of scalar fields to be displayed, it can be seen that the magnitude and components of vector fields are available by default, *e.g.* displayed as **U:X**, so that it was not necessary to create **Ux** using **postProcess**. Nevertheless, the user should *deselect* all series except **Ux** (or **U:x**). A square colour box in the adjacent column to the selected series indicates the line colour. The user can edit this most easily by a double click of the mouse over that selection.

In order to format the graph, the user should modify the settings below the **Line Series** panel, namely **Line Color**, **Line Thickness**, **Line Style**, **Marker Style** and **Chart Axes**.

Also the user can click one of the buttons above the top left corner of the **XY Plot**. The third button, for example, allows the user to control **View Settings** in which the user can set title and legend for each axis, for example. Also, the user can set font, colour and alignment of the axes titles, and has several options for axis range and labels in linear or logarithmic scales.

Figure 2.11 is a graph produced using **ParaView**. The user can produce a graph however he/she wishes. For information, the graph in Figure 2.11 was produced with the options for axes of: **Standard** type of Notation; **Specify Axis Range** selected; titles in **Sans Serif 12** font. The graph is displayed as a set of points rather than a line by activating the **Enable Line Series** button in the **Display** window. Note: if this button appears to be inactive by being “greyed out”, it can be made active by selecting and *deselecting* the sets of variables in the **Line Series** panel. Once the **Enable Line Series** button is selected, the **Line Style** and **Marker Style** can be adjusted to the user’s preference.

## 2.1.6 Introducing mesh grading

The error in any solution will be more pronounced in regions where the form of the true solution differ widely from the form assumed in the chosen numerical schemes. For example a numerical scheme based on linear variations of variables over cells can only generate an exact solution if the true solution is itself linear in form. The error is largest



in regions where the true solution deviates greatest from linear form, *i.e.* where the change in gradient is largest. Error decreases with cell size.

It is useful to have an intuitive appreciation of the form of the solution before setting up any problem. It is then possible to anticipate where the errors will be largest and to grade the mesh so that the smallest cells are in these regions. In the `cavity` case the large variations in velocity can be expected near a wall and so in this part of the tutorial the mesh will be graded to be smaller in this region. By using the same number of cells, greater accuracy can be achieved without a significant increase in computational cost.

A mesh of  $20 \times 20$  cells with grading towards the walls will be created for the lid-driven cavity problem and the results from the finer mesh of section 2.1.5.2 will then be mapped onto the graded mesh to use as an initial condition. The results from the graded mesh will be compared with those from the previous meshes. Since the changes to the `blockMeshDict` dictionary are fairly substantial, the case used for this part of the tutorial, `cavityGrade`, is supplied in the `$FOAM_RUN/tutorials/incompressible/icoFoam/cavity` directory.

### 2.1.6.1 Creating the graded mesh

The mesh now needs 4 blocks as different mesh grading is needed on the left and right and top and bottom of the domain. The block structure for this mesh is shown in Figure 2.12. The user can view the `blockMeshDict` file in the `system` subdirectory of `cavityGrade`; for completeness the key elements of the `blockMeshDict` file are also reproduced below. Each block now has 10 cells in the  $x$  and  $y$  directions and the ratio between largest and smallest cells is 2.

```

17  scale    0.1;
18
19  vertices
20  (
21      (0 0 0)
22      (0.5 0 0)
23      (1 0 0)
24      (0 0.5 0)
25      (0.5 0.5 0)
26      (1 0.5 0)
27      (0 1 0)
28      (0.5 1 0)
29      (1 1 0)
30      (0 0 0.1)
31      (0.5 0 0.1)
32      (1 0 0.1)
33      (0 0.5 0.1)
34      (0.5 0.5 0.1)
35      (1 0.5 0.1)
36      (0 1 0.1)
37      (0.5 1 0.1)
38      (1 1 0.1)
39  );
40
41  blocks
42  (
43      hex (0 1 4 3 9 10 13 12) (10 10 1) simpleGrading (2 2 1)
44      hex (1 2 5 4 10 11 14 13) (10 10 1) simpleGrading (0.5 2 1)
45      hex (3 4 7 6 12 13 16 15) (10 10 1) simpleGrading (2 0.5 1)
46      hex (4 5 8 7 13 14 17 16) (10 10 1) simpleGrading (0.5 0.5 1)
47  );
48
49  edges
50  (
51  );
52
53  boundary
54  (
55      movingWall
56      {
57          type wall;
58          faces
59          (
60              (6 15 16 7)
61              (7 16 17 8)
62          );
63      }
64  );

```

```

63     }
64     fixedWalls
65     {
66         type wall;
67         faces
68         (
69             (3 12 15 6)
70             (0 9 12 3)
71             (0 1 10 9)
72             (1 2 11 10)
73             (2 5 14 11)
74             (5 8 17 14)
75         );
76     }
77     frontAndBack
78     {
79         type empty;
80         faces
81         (
82             (0 3 4 1)
83             (1 4 5 2)
84             (3 6 7 4)
85             (4 7 8 5)
86             (9 10 13 12)
87             (10 11 14 13)
88             (12 13 16 15)
89             (13 14 17 16)
90         );
91     }
92 );
93
94
95 // *****

```

Once familiar with the *blockMeshDict* file for this case, the user can execute **blockMesh** from the command line. The graded mesh can be viewed as before using **paraFoam** as described in section 2.1.2.

### 2.1.6.2 Changing time and time step

The highest velocities and smallest cells are next to the lid, therefore the highest Courant number will be generated next to the lid, for reasons given in section 2.1.1.4. It is therefore useful to estimate the size of the cells next to the lid to calculate an appropriate time step for this case.

When a nonuniform mesh grading is used, **blockMesh** calculates the cell sizes using a geometric progression. Along a length  $l$ , if  $n$  cells are requested with a ratio of  $R$  between the last and first cells, the size of the smallest cell,  $\delta x_s$ , is given by:

$$\delta x_s = l \frac{r - 1}{\alpha r - 1} \quad (2.5)$$

where  $r$  is the ratio between one cell size and the next which is given by:

$$r = R^{\frac{1}{n-1}} \quad (2.6)$$

and

$$\alpha = \begin{cases} R & \text{for } R > 1, \\ 1 - r^{-n} + r^{-1} & \text{for } R < 1. \end{cases} \quad (2.7)$$

For the **cavityGrade** case the number of cells in each direction in a block is 10, the ratio between largest and smallest cells is 2 and the block height and width is 0.05 m. Therefore the smallest cell length is 3.45 mm. From Equation 2.2, the time step should be less than 3.45 ms to maintain a Courant of less than 1. To ensure that results are written out at convenient time intervals, the time step **deltaT** should be reduced to 2.5 ms and the **writeInterval** set to 40 so that results are written out every 0.1 s. These settings can be viewed in the *cavityGrade/system/controlDict* file.

The `startTime` needs to be set to that of the final conditions of the case `cavityFine`, *i.e.* 0.7. Since `cavity` and `cavityFine` converged well within the prescribed run time, we can set the run time for case `cavityGrade` to 0.1 s, *i.e.* the `endTime` should be 0.8.

### 2.1.6.3 Mapping fields

As in section 2.1.5.3, use `mapFields` to map the final results from case `cavityFine` onto the mesh for case `cavityGrade`. Enter the `cavityGrade` directory and execute `mapFields` by:

```
cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity/cavityGrade
mapFields ../cavityFine -consistent
```

Now run `icoFoam` from the case directory and monitor the run time information. View the converged results for this case and compare with other results using post-processing tools described previously in section 2.1.5.6 and section 2.1.5.7.

## 2.1.7 Increasing the Reynolds number

The cases solved so far have had a Reynolds number of 10. This is very low and leads to a stable solution quickly with only small secondary vortices at the bottom corners of the cavity. We will now increase the Reynolds number to 100, at which point the solution takes a noticeably longer time to converge. The coarsest mesh in case `cavity` will be used initially. The user should make a copy of the `cavity` case and name it `cavityHighRe` by typing:

```
cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity
cp -r cavity cavityHighRe
```

### 2.1.7.1 Pre-processing

Enter the `cavityHighRe` case and edit the *transportProperties* dictionary. Since the Reynolds number is required to be increased by a factor of 10, decrease the kinematic viscosity by a factor of 10, *i.e.* to  $1 \times 10^{-3} \text{ m}^2 \text{ s}^{-1}$ . We can now run this case by restarting from the solution at the end of the `cavity` case run. To do this we can use the option of setting the `startFrom` keyword to `latestTime` so that `icoFoam` takes as its initial data the values stored in the directory corresponding to the most recent time, *i.e.* 0.5. The `endTime` should be set to 2 s.

### 2.1.7.2 Running the code

Run `icoFoam` for this case from the case directory and view the run time information. When running a job in the background, the following UNIX commands can be useful:

`nohup` enables a command to keep running after the user who issues the command has logged out;

`nice` changes the priority of the job in the kernel's scheduler; a niceness of -20 is the highest priority and 19 is the lowest priority.

This is useful, for example, if a user wishes to set a case running on a remote machine and does not wish to monitor it heavily, in which case they may wish to give it low priority on the machine. In that case the `nohup` command allows the user to log out of a

remote machine he/she is running on and the job continues running, while `nice` can set the priority to 19. For our case of interest, we can execute the command in this manner as follows:

```
cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity/cavityHighRe
nohup nice -n 19 icoFoam > log &
cat log
```

In previous runs you may have noticed that `icoFoam` stops solving for velocity `U` quite quickly but continues solving for pressure `p` for a lot longer or until the end of the run. In practice, once `icoFoam` stops solving for `U` and the initial residual of `p` is less than the tolerance set in the `fvSolution` dictionary (typically  $10^{-6}$ ), the run has effectively converged and can be stopped once the field data has been written out to a time directory. For example, at convergence a sample of the `log` file from the run on the `cavityHighRe` case appears as follows in which the velocity has already converged after 1.62 s and initial pressure residuals are small; `No Iterations 0` indicates that the solution of `U` has stopped:

```
1
2 Time = 1.63
3
4 Courant Number mean: 0.221985 max: 0.839923
5 smoothSolver: Solving for Ux, Initial residual = 3.64032e-06, Final residual = 3.64032e-06, No Iterations 0
6 smoothSolver: Solving for Uy, Initial residual = 4.20677e-06, Final residual = 4.20677e-06, No Iterations 0
7 DICPCG: Solving for p, Initial residual = 2.11678e-06, Final residual = 7.25303e-07, No Iterations 3
8 time step continuity errors : sum local = 7.25166e-09, global = 4.96308e-19, cumulative = -1.28342e-17
9 DICPCG: Solving for p, Initial residual = 1.36075e-06, Final residual = 7.94478e-07, No Iterations 1
10 time step continuity errors : sum local = 7.77548e-09, global = -4.78772e-19, cumulative = -1.3313e-17
11 ExecutionTime = 0.38 s ClockTime = 0 s
12
13 Time = 1.635
14
15 Courant Number mean: 0.221986 max: 0.839923
16 smoothSolver: Solving for Ux, Initial residual = 3.56036e-06, Final residual = 3.56036e-06, No Iterations 0
17 smoothSolver: Solving for Uy, Initial residual = 4.11726e-06, Final residual = 4.11726e-06, No Iterations 0
18 DICPCG: Solving for p, Initial residual = 2.03881e-06, Final residual = 8.18692e-07, No Iterations 3
19 time step continuity errors : sum local = 8.38471e-09, global = -6.27334e-19, cumulative = -1.39403e-17
20 DICPCG: Solving for p, Initial residual = 1.36655e-06, Final residual = 7.94623e-07, No Iterations 1
21 time step continuity errors : sum local = 8.25673e-09, global = 5.87298e-20, cumulative = -1.38816e-17
22 ExecutionTime = 0.38 s ClockTime = 0 s
```

## 2.1.8 High Reynolds number flow

View the results in `paraFoam` and display the velocity vectors. The secondary vortices in the corners have increased in size somewhat. The user can then increase the Reynolds number further by decreasing the viscosity and then rerun the case. The number of vortices increases so the mesh resolution around them will need to increase in order to resolve the more complicated flow patterns. In addition, as the Reynolds number increases the time to convergence increases. The user should monitor residuals and extend the `endTime` accordingly to ensure convergence.

The need to increase spatial and temporal resolution then becomes impractical as the flow moves into the turbulent regime, where problems of solution stability may also occur. Of course, many engineering problems have very high Reynolds numbers and it is infeasible to bear the huge cost of solving the turbulent behaviour directly. Instead Reynolds-averaged simulation (RAS) turbulence models are used to solve for the mean flow behaviour and calculate the statistics of the fluctuations. The standard  $k - \varepsilon$  model with wall functions will be used in this tutorial to solve the lid-driven cavity case with a Reynolds number of  $10^4$ . Two extra variables are solved for:  $k$ , the turbulent kinetic energy; and,  $\varepsilon$ , the turbulent dissipation rate. The additional equations and models for turbulent flow are implemented into a OpenFOAM solver called `pisoFoam`.

### 2.1.8.1 Pre-processing

Change directory to the `cavity` case in the `$FOAM_RUN/tutorials/incompressible/pisoFoam/-RAS` directory (N.B: the **pisoFoam/RAS** directory). Generate the mesh by running `blockMesh` as before. Mesh grading towards the wall is not necessary when using the standard  $k - \varepsilon$  model with wall functions since the flow in the near wall cell is modelled, rather than having to be resolved.

A range of wall function models is available in OpenFOAM that are applied as boundary conditions on individual patches. This enables different wall function models to be applied to different wall regions. The choice of wall function models are specified through the turbulent viscosity field,  $\nu_t$  in the `0/nut` file:

```

17 dimensions      [0 2 -1 0 0 0 0];
18
19 internalField    uniform 0;
20
21 boundaryField
22 {
23     "(movingWall|fixedWalls)"
24     {
25         type      nutkWallFunction;
26         value      uniform 0;
27     }
28
29     frontAndBack
30     {
31         type      empty;
32     }
33 }
34
35
36 // ***** //
```

This case uses standard wall functions, specified by the `nutWallFunction` keyword entry on the `movingWall` and `fixedWalls` patches. Other wall function models include the rough wall functions, specified though the `nutRoughWallFunction` keyword.

The user should now open the field files for  $k$  and  $\varepsilon$  (`0/k` and `0/epsilon`) and examine their boundary conditions. For a wall boundary condition `wall`,  $\varepsilon$  is assigned a `epsilonWallFunction` boundary condition and a `kqRwallFunction` boundary condition is assigned to  $k$ . The latter is a generic boundary condition that can be applied to any field that are of a turbulent kinetic energy type, *e.g.*  $k$ ,  $q$  or Reynolds Stress  $R$ . The initial values for  $k$  and  $\varepsilon$  are set using an estimated fluctuating component of velocity  $\mathbf{U}'$  and a turbulent length scale,  $l$ .  $k$  and  $\varepsilon$  are defined in terms of these parameters as follows:

$$k = \frac{1}{2} \overline{\mathbf{U}' \cdot \mathbf{U}'} \quad (2.8)$$

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{l} \quad (2.9)$$

where  $C_\mu$  is a constant of the  $k - \varepsilon$  model equal to 0.09. For a Cartesian coordinate system,  $k$  is given by:

$$k = \frac{1}{2} (U_x'^2 + U_y'^2 + U_z'^2) \quad (2.10)$$

where  $U_x'^2$ ,  $U_y'^2$  and  $U_z'^2$  are the fluctuating components of velocity in the  $x$ ,  $y$  and  $z$  directions respectively. Let us assume the initial turbulence is isotropic, *i.e.*  $U_x'^2 = U_y'^2 = U_z'^2$ , and equal to 5% of the lid velocity and that  $l$ , is equal to 20% of the box width, 0.1

m, then  $k$  and  $\varepsilon$  are given by:

$$U'_x = U'_y = U'_z = \frac{5}{100} 1 \text{ m s}^{-1} \quad (2.11)$$

$$\Rightarrow k = \frac{3}{2} \left( \frac{5}{100} \right)^2 \text{ m}^2 \text{ s}^{-2} = 3.75 \times 10^{-3} \text{ m}^2 \text{ s}^{-2} \quad (2.12)$$

$$\varepsilon = \frac{0.09^{0.75} \times (3.75 \times 10^{-3})^{1.5}}{0.2 \times 0.1} \approx 1.89 \times 10^{-3} \text{ m}^2 \text{ s}^{-3} \quad (2.13)$$

These form the initial conditions for  $k$  and  $\varepsilon$ . The initial conditions for  $\mathbf{U}$  and  $p$  are  $(0, 0, 0)$  and  $0$  respectively as before.

Turbulence modelling includes a range of methods, *e.g.* RAS or large-eddy simulation (LES), that are provided in OpenFOAM. In most transient solvers, the choice of turbulence modelling method is selectable at run-time through the `simulationType` keyword in `turbulenceProperties` dictionary. The user can view this file in the `constant` directory:

```

17 simulationType      RAS;
18
19 RAS
20 {
21     RASModel         kEpsilon;
22
23     turbulence        on;
24
25     printCoeffs       on;
26 }
27
28
29 // ***** //
```

The options for `simulationType` are `laminar`, `RAS` and `LES`. More information on turbulence models can be found in the [Extended Code Guide](#). With `RAS` selected in this case, the choice of `RAS` modelling is specified in a `turbulenceProperties` subdictionary, also in the `constant` directory. The turbulence model is selected by the `RASModel` entry from a long list of available models that are listed in User Guide Table A.5. The `kEpsilon` model should be selected which is the standard  $k - \varepsilon$  model; the user should also ensure that `turbulence` calculation is switched `on`.

The coefficients for each turbulence model are stored within the respective code with a set of default values. Setting the optional switch called `printCoeffs` to `on` will make the default values be printed to standard output, *i.e.* the terminal, when the model is called at run time. The coefficients are printed out as a sub-dictionary whose name is that of the model name with the word `Coeffs` appended, *e.g.* `kEpsilonCoeffs` in the case of the `kEpsilon` model. The coefficients of the model, *e.g.* `kEpsilon`, can be modified by optionally including (copying and pasting) that sub-dictionary within the `turbulenceProperties` file and adjusting values accordingly.

The user should next set the laminar kinematic viscosity in the `transportProperties` dictionary. To achieve a Reynolds number of  $10^4$ , a kinematic viscosity of  $10^{-5} \text{ m}^2 \text{ s}^{-1}$  is required based on the Reynolds number definition given in Equation 2.1.

Finally the user should set the `startTime`, `stopTime`, `deltaT` and the `writeInterval` in the `controlDict`. Set `deltaT` to  $0.005 \text{ s}$  to satisfy the Courant number restriction and the `endTime` to  $10 \text{ s}$ .

### 2.1.8.2 Running the code

Execute `pisoFoam` by entering the case directory and typing “`pisoFoam`” in a terminal. In this case, where the viscosity is low, the boundary layer next to the moving lid is very thin and the cells next to the lid are comparatively large so the velocity at their

centres are much less than the lid velocity. In fact, after  $\approx 100$  time steps it becomes apparent that the velocity in the cells adjacent to the lid reaches an upper limit of around  $0.2 \text{ m s}^{-1}$  hence the maximum Courant number does not rise much above 0.2. It is sensible to increase the solution time by increasing the time step to a level where the Courant number is much closer to 1. Therefore reset `deltaT` to 0.02 s and, on this occasion, set `startFrom` to `latestTime`. This instructs `pisoFoam` to read the start data from the latest time directory, *i.e.* `10.0`. The `endTime` should be set to 20 s since the run converges a lot slower than the laminar case. Restart the run as before and monitor the convergence of the solution. View the results at consecutive time steps as the solution progresses to see if the solution converges to a steady-state or perhaps reaches some periodically oscillating state. In the latter case, convergence may never occur but this does not mean the results are inaccurate.

### 2.1.9 Changing the case geometry

A user may wish to make changes to the geometry of a case and perform a new simulation. It may be useful to retain some or all of the original solution as the starting conditions for the new simulation. This is a little complex because the fields of the original solution are not consistent with the fields of the new case. However the `mapFields` utility can map fields that are inconsistent, either in terms of geometry or boundary types or both.

As an example, let us go to the `cavityClipped` case in the `icoFoam` directory which consists of the standard `cavity` geometry but with a square of length 0.04 m removed from the bottom right of the cavity, according to the `blockMeshDict` below:

```

17  scale    0.1;
18
19  vertices
20  (
21      (0 0 0)
22      (0.6 0 0)
23      (0 0.4 0)
24      (0.6 0.4 0)
25      (1 0.4 0)
26      (0 1 0)
27      (0.6 1 0)
28      (1 1 0)
29
30      (0 0 0.1)
31      (0.6 0 0.1)
32      (0 0.4 0.1)
33      (0.6 0.4 0.1)
34      (1 0.4 0.1)
35      (0 1 0.1)
36      (0.6 1 0.1)
37      (1 1 0.1)
38  );
39
40  blocks
41  (
42      hex (0 1 3 2 8 9 11 10) (12 8 1) simpleGrading (1 1 1)
43      hex (2 3 6 5 10 11 14 13) (12 12 1) simpleGrading (1 1 1)
44      hex (3 4 7 6 11 12 15 14) (8 12 1) simpleGrading (1 1 1)
45  );
46
47  edges
48  (
49  );
50
51  boundary
52  (
53      lid
54      {
55          type wall;
56          faces
57          (
58              (5 13 14 6)
59              (6 14 15 7)
60          );
61      }
62      fixedWalls

```



```

63     {
64         type wall;
65         faces
66         (
67             (0 8 10 2)
68             (2 10 13 5)
69             (7 15 12 4)
70             (4 12 11 3)
71             (3 11 9 1)
72             (1 9 8 0)
73         );
74     }
75     frontAndBack
76     {
77         type empty;
78         faces
79         (
80             (0 2 3 1)
81             (2 5 6 3)
82             (3 6 7 4)
83             (8 9 11 10)
84             (10 11 14 13)
85             (11 12 15 14)
86         );
87     }
88 );
89
90
91 // *****

```

Generate the mesh with `blockMesh`. The patches are set as according to the previous cavity cases. For the sake of clarity in describing the field mapping process, the upper wall patch is renamed `lid`, previously the `movingWall` patch of the original `cavity`.

In an inconsistent mapping, there is no guarantee that all the field data can be mapped from the source case. The remaining data must come from field files in the target case itself. Therefore field data must exist in the time directory of the target case before mapping takes place. In the `cavityClipped` case the mapping is set to occur at time 0.5 s, since the `startTime` is set to 0.5 s in the `controlDict`. Therefore the user needs to copy initial field data to that directory, *e.g.* from time 0:

```

cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity/cavityClipped
cp -r 0 0.5

```

Before mapping the data, the user should view the geometry and fields at 0.5 s.

Now we wish to map the velocity and pressure fields from `cavity` onto the new fields of `cavityClipped`. Since the mapping is inconsistent, we need to edit the `mapFieldsDict` dictionary, located in the `system` directory. The dictionary contains 2 keyword entries: `patchMap` and `cuttingPatches`. The `patchMap` list contains a mapping of patches from the source fields to the target fields. It is used if the user wishes a patch in the target field to inherit values from a corresponding patch in the source field. In `cavityClipped`, we wish to inherit the boundary values on the `lid` patch from `movingWall` in `cavity` so we must set the `patchMap` as:

```

patchMap
(
    lid movingWall
);

```

The `cuttingPatches` list contains names of target patches whose values are to be mapped from the source internal field through which the target patch cuts. In this case we will include the `fixedWalls` to demonstrate the interpolation process.

```

cuttingPatches

```



```
(  
    fixedWalls  
);
```

Now the user should run `mapFields`, from within the *cavityClipped* directory:

```
mapFields ../cavity
```

The user can view the mapped field as shown in Figure 2.13. The boundary patches have inherited values from the source case as we expected. Having demonstrated this, however, we actually wish to reset the velocity on the `fixedWalls` patch to  $(0, 0, 0)$ . Edit the `U` field, go to the `fixedWalls` patch and change the field from `nonuniform` to `uniform`  $(0, 0, 0)$ . The `nonuniform` field is a list of values that requires deleting in its entirety. Now run the case with `icoFoam`.

### 2.1.10 Post-processing the modified geometry

Velocity glyphs can be generated for the case as normal, first at time 0.5 s and later at time 0.6 s, to compare the initial and final solutions. In addition, we provide an outline of the geometry which requires some care to generate for a 2D case. The user should select **Extract Block** from the **Filter** menu and, in the **Parameter** panel, highlight the patches of interest, namely the lid and `fixedWalls`. On clicking **Apply**, these items of geometry can be displayed by selecting **Wireframe** in the **Properties** panel. Figure 2.14 displays the patches in black and shows vortices forming in the bottom corners of the modified geometry.

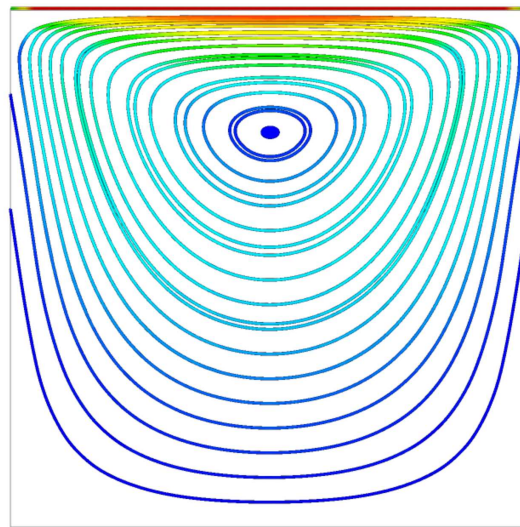


Figure 2.9: Streamlines in the cavity case.

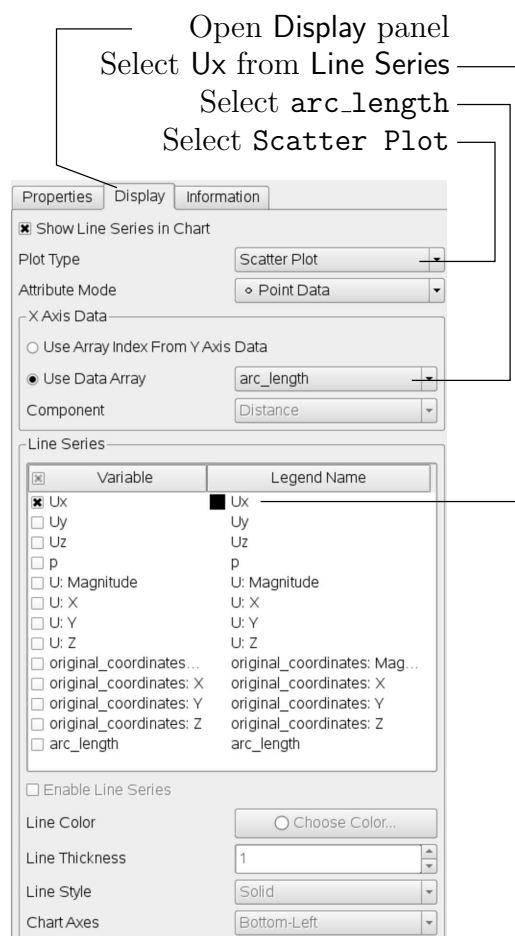


Figure 2.10: Selecting fields for graph plotting.

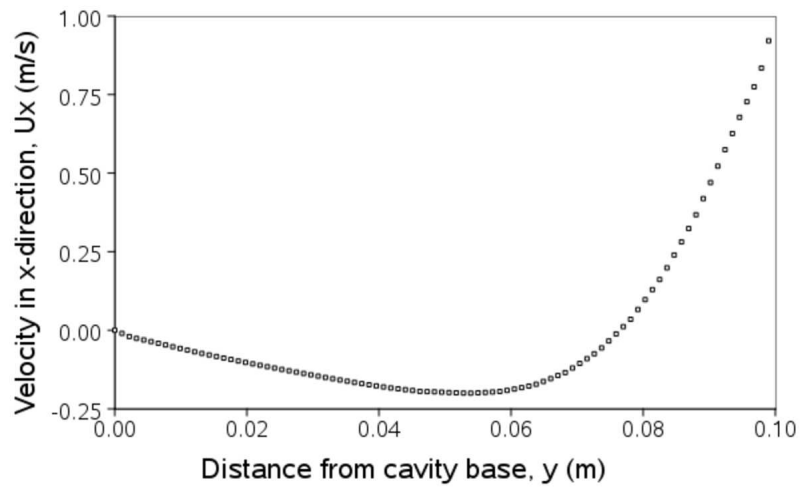


Figure 2.11: Plotting graphs in ParaView.

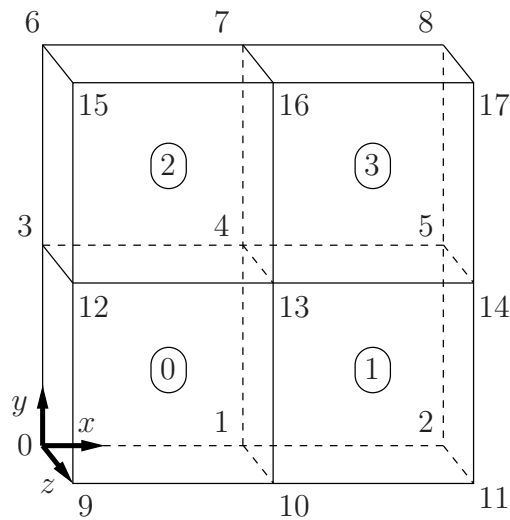


Figure 2.12: Block structure of the graded mesh for the cavity (block numbers encircled).

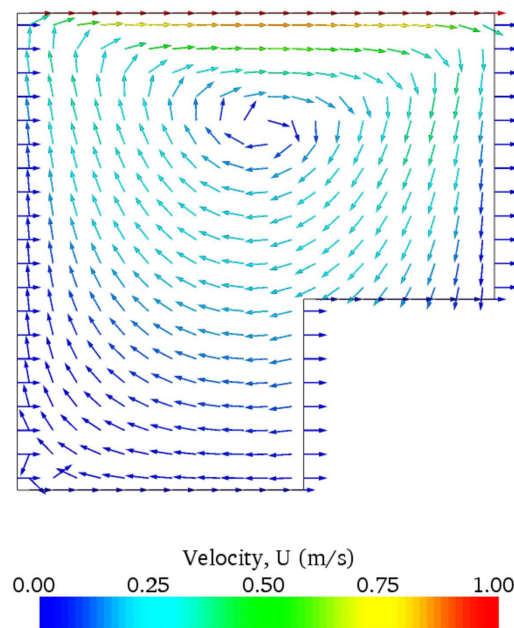


Figure 2.13: cavity solution velocity field mapped onto cavityClipped.

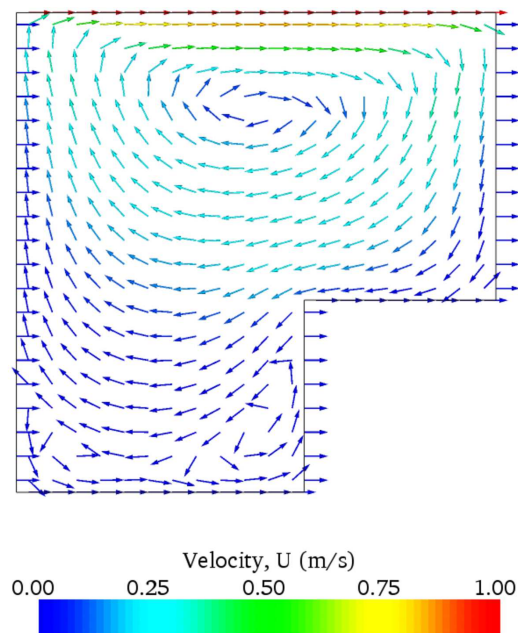


Figure 2.14: cavityClipped solution for velocity field.

## 2.2 Flow around a cylinder

Tutorial path:

- [\\$FOAM\\_TUTORIALS/basic/potentialFoam/cylinder](#)

In this example we shall investigate potential flow around a cylinder using the `potentialFoam` solver. This example introduces the following OpenFOAM features:

- non-orthogonal meshes;
- generating an analytical solution to a problem in OpenFOAM;
- use of a dynamic code to generate the block vertices;
- use of a coded function object to compare results against the analytical solution.

### 2.2.1 Problem specification

The problem is defined as follows:

***Solution domain*** The domain is 2 dimensional and consists of a square domain with a cylinder collocated with the centre of the square as shown in Figure 2.15.

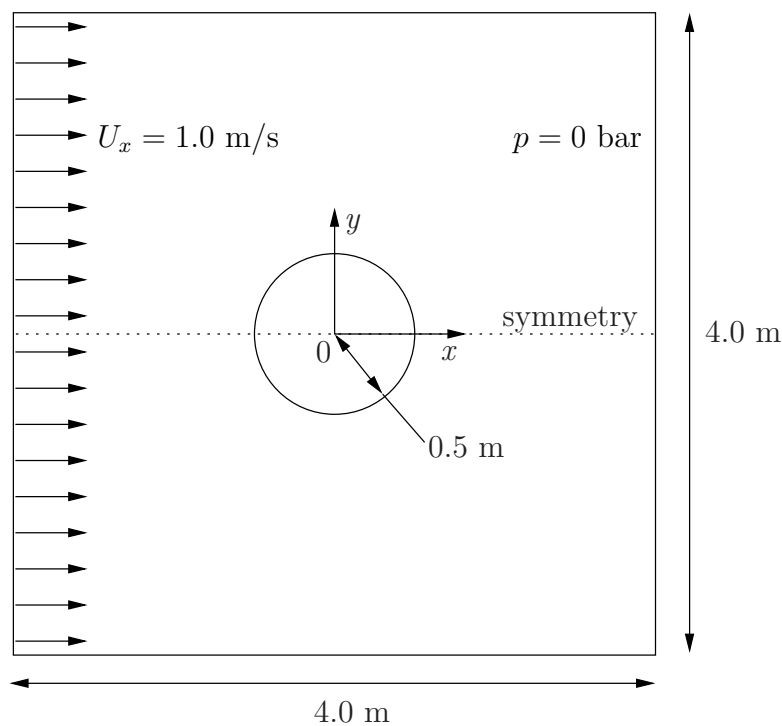


Figure 2.15: Geometry of flow round a cylinder

#### *Governing equations*

- Mass continuity for an incompressible fluid

$$\nabla \cdot \mathbf{U} = 0 \quad (2.14)$$

- Pressure equation for an incompressible, irrotational fluid assuming steady-state conditions

$$\nabla^2 p = 0 \quad (2.15)$$

### Boundary conditions

- Inlet (left) with fixed velocity  $\mathbf{U} = (1, 0, 0)$  m/s.
- Outlet (right) with a fixed pressure  $p = 0$  Pa.
- No-slip wall (bottom);
- Symmetry plane (top).

**Initial conditions**  $U = 0$  m/s,  $p = 0$  Pa — required in OpenFOAM input files but not necessary for the solution since the problem is steady-state.

**Solver name** `potentialFoam`: a potential flow code, *i.e.* assumes the flow is incompressible, steady, irrotational, inviscid and it ignores gravity.

**Case name** `cylinder` case located in the `$FOAM_TUTORIALS/basic/potentialFoam` directory.

### 2.2.2 Note on potentialFoam

`potentialFoam` is a useful solver to validate OpenFOAM since the assumptions of potential flow are such that an analytical solution exists for cases whose geometries are relatively simple. In this example of flow around a cylinder an analytical solution exists with which we can compare our numerical solution. `potentialFoam` can also be run more like a utility to provide a (reasonably) conservative initial  $\mathbf{U}$  field for a problem. When running certain cases, this can be useful for avoiding instabilities due to the initial field being unstable. In short, `potentialFoam` creates a conservative field from a non-conservative initial field supplied by the user.

### 2.2.3 Mesh generation

Mesh generation using `blockMesh` has been described in tutorials in the User Guide. In this case, the mesh consists of 10 blocks as shown in Figure 2.16. Remember that all

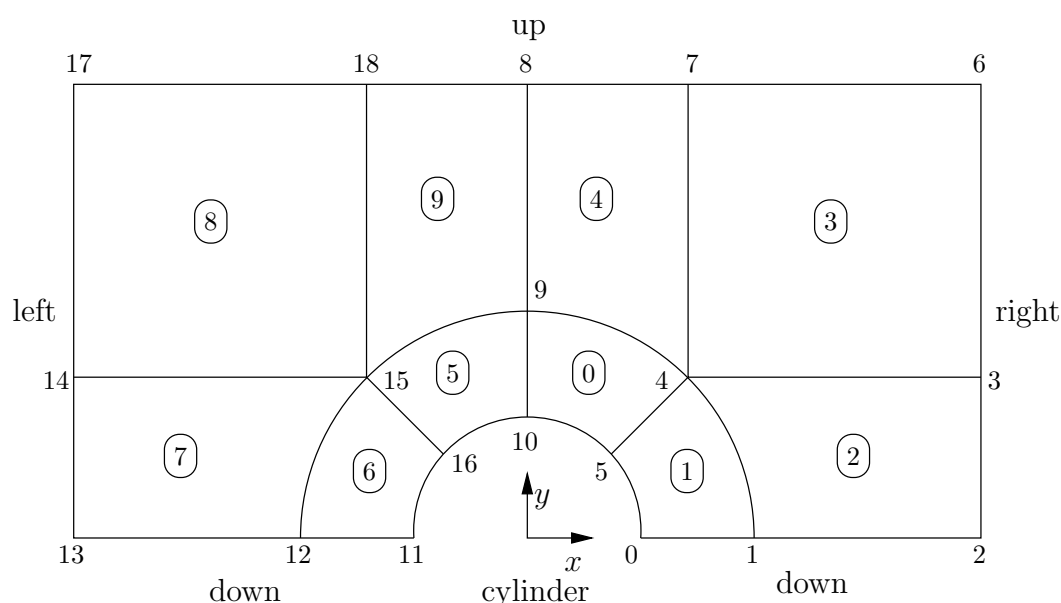


Figure 2.16: Blocks in cylinder geometry

meshes are treated as 3 dimensional in OpenFOAM. If we wish to solve a 2 dimensional

problem, we must describe a 3 dimensional mesh that is only one cell thick in the third direction that is not solved. In Figure 2.16 we show only the back plane of the geometry, along  $z = -0.5$ , in which the vertex numbers are numbered 0-18. The other 19 vertices in the front plane,  $z = +0.5$ , are numbered in the same order as the back plane, as shown in the mesh description file below:

```

1  /*-----* C++ *-----*/
2  |=====|
3  |  \  /  | F ield      | OpenFOAM: The Open Source CFD Toolbox
4  |  \  /  | O peration  | Version: v2506
5  |  \  /  | A nd        | Website: www.openfoam.com
6  |  \  /  | M anipulation|
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        blockMeshDict;
14 }
15 // * * * * *
16
17 scale 1;
18
19 // Geometric parameters
20 rInner 0.5;
21 rOuter 1;
22 xmax 2;
23 ymax 2;
24
25 zmin -0.5; // Back/front locations
26 zmax 0.5;
27
28 // Divisions: Radial, quarter circumference, outer region and z-directions.
29 nRadial 10;
30 nQuarter 10;
31 nxOuter 20;
32 nyOuter 20;
33 nz 1;
34
35 // -----
36
37 // Derived quantities
38 rInner45 ${{ $rInner * sqrt(0.5) }};
39 rOuter45 ${{ $rOuter * sqrt(0.5) }};
40
41 vertices #codeStream
42 {
43     codeInclude
44     #{
45         #include "pointField.H"
46     };
47
48     code
49     #{
50         pointField points
51         ({
52             /* 0*/ { $rInner, 0, $zmin },
53             /* 1*/ { $rOuter, 0, $zmin },
54             /* 2*/ { $xmax, 0, $zmin },
55             /* 3*/ { $xmax, $rOuter45, $zmin },
56             /* 4*/ { $rOuter45, $rOuter45, $zmin },
57             /* 5*/ { $rInner45, $rInner45, $zmin },
58             /* 6*/ { $xmax, $ymax, $zmin },
59             /* 7*/ { $rOuter45, $ymax, $zmin },
60             /* 8*/ { 0, $ymax, $zmin },
61             /* 9*/ { 0, $rOuter, $zmin },
62             /*10*/ { 0, $rInner, $zmin },
63             /*11*/ { -$rInner, 0, $zmin },
64             /*12*/ { -$rOuter, 0, $zmin },
65             /*13*/ { -$xmax, 0, $zmin },
66             /*14*/ { -$xmax, $rOuter45, $zmin },
67             /*15*/ { -$rOuter45, $rOuter45, $zmin },
68             /*16*/ { -$rInner45, $rInner45, $zmin },
69             /*17*/ { -$xmax, $ymax, $zmin },
70             /*18*/ { -$rOuter45, $ymax, $zmin }
71         });
72
73         // Duplicate z points for zmax
74         const label sz = points.size();
75         points.resize(2*sz);

```

```

76         for (label i = 0; i < sz; ++i)
77         {
78             const point& pt = points[i];
79             points[i + sz] = point(pt.x(), pt.y(), $zmax);
80         }
81
82         os << points;
83     #};
84 };
85
86 // Can remove unneeded variables
87 #remove ( "r(Inner|Outer).*" "[xy](min|max)" )
88
89 blocks
90 (
91     hex (5 4 9 10 24 23 28 29) ($nRadial $nQuarter $nz) grading (1 1 1)
92     hex (0 1 4 5 19 20 23 24) ($nRadial $nQuarter $nz) grading (1 1 1)
93     hex (1 2 3 4 20 21 22 23) ($nxOuter $nQuarter $nz) grading (1 1 1)
94     hex (4 3 6 7 23 22 25 26) ($nxOuter $nyOuter $nz) grading (1 1 1)
95     hex (9 4 7 8 28 23 26 27) ($nQuarter $nyOuter $nz) grading (1 1 1)
96     hex (15 16 10 9 34 35 29 28) ($nRadial $nQuarter $nz) grading (1 1 1)
97     hex (12 11 16 15 31 30 35 34) ($nRadial $nQuarter $nz) grading (1 1 1)
98     hex (13 12 15 14 32 31 34 33) ($nxOuter $nQuarter $nz) grading (1 1 1)
99     hex (14 15 18 17 33 34 37 36) ($nxOuter $nyOuter $nz) grading (1 1 1)
100    hex (15 9 8 18 34 28 27 37) ($nQuarter $nyOuter $nz) grading (1 1 1)
101 );
102
103 edges
104 (
105     // Inner cylinder
106     arc 0 5 origin (0 0 $zmin)
107     arc 5 10 origin (0 0 $zmin)
108     arc 1 4 origin (0 0 $zmin)
109     arc 4 9 origin (0 0 $zmin)
110     arc 19 24 origin (0 0 $zmax)
111     arc 24 29 origin (0 0 $zmax)
112     arc 20 23 origin (0 0 $zmax)
113     arc 23 28 origin (0 0 $zmax)
114     // Intermediate cylinder
115     arc 11 16 origin (0 0 $zmin)
116     arc 16 10 origin (0 0 $zmin)
117     arc 12 15 origin (0 0 $zmin)
118     arc 15 9 origin (0 0 $zmin)
119     arc 30 35 origin (0 0 $zmax)
120     arc 35 29 origin (0 0 $zmax)
121     arc 31 34 origin (0 0 $zmax)
122     arc 34 28 origin (0 0 $zmax)
123 );
124
125 boundary
126 (
127     down
128     {
129         type symmetryPlane;
130         faces
131         (
132             (0 1 20 19)
133             (1 2 21 20)
134             (12 11 30 31)
135             (13 12 31 32)
136         );
137     }
138     right
139     {
140         type patch;
141         faces
142         (
143             (2 3 22 21)
144             (3 6 25 22)
145         );
146     }
147     up
148     {
149         type symmetryPlane;
150         faces
151         (
152             (7 8 27 26)
153             (6 7 26 25)
154             (8 18 37 27)

```



```

155         (18 17 36 37)
156     );
157 }
158 left
159 {
160     type patch;
161     faces
162     (
163         (14 13 32 33)
164         (17 14 33 36)
165     );
166 }
167 cylinder
168 {
169     type symmetry;
170     faces
171     (
172         (10 5 24 29)
173         (5 0 19 24)
174         (16 10 29 35)
175         (11 16 35 30)
176     );
177 }
178 );
179
180 mergePatchPairs
181 (
182 );
183
184
185 // *****

```

## 2.2.4 Boundary conditions and initial fields

Edit the case files to set the boundary conditions in accordance with the problem description in Figure 2.15, *i.e.* the left boundary should be an **Inlet**, the right boundary should be an **Outlet** and the down and cylinder boundaries should be **symmetryPlane**. The top boundary conditions is chosen so that we can make the most genuine comparison with our analytical solution which uses the assumption that the domain is infinite in the  $y$  direction. The result is that the normal gradient of  $\mathbf{U}$  is small along a plane coinciding with our boundary. We therefore impose the condition that the normal component is zero, *i.e.* specify the boundary as a **symmetryPlane**, thereby ensuring that the comparison with the analytical is reasonable.

## 2.2.5 Running the case

No fluid properties need be specified in this problem since the flow is assumed to be incompressible and inviscid. In the **system** subdirectory, the **controlDict** specifies the control parameters for the run. Note that since we assume steady flow, we only run for 1 time step:

```

1  /*-----* C++ *-----*/
2  |=====|
3  | \      / F i e l d      | OpenFOAM: The Open Source CFD Toolbox
4  |  \    / O peration     | Version: v2506
5  |   \  / A nd            | Website: www.openfoam.com
6  |    \/ M anipulation    |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        dictionary;
13     object       controlDict;
14 }
15 // *****
16
17 application      potentialFoam;
18
19 startFrom        latestTime;
20

```

```

21  startTime      0;
22
23  stopAt         nextWrite;
24
25  endTime        1;
26
27  deltaT         1;
28
29  writeControl    timeStep;
30
31  writeInterval   1;
32
33  purgeWrite     0;
34
35  writeFormat     ascii;
36
37  writePrecision  6;
38
39  writeCompression off;
40
41  timeFormat      general;
42
43  timePrecision   6;
44
45  runTimeModifiable true;
46
47  functions
48  {
49      error
50      {
51          name      error;
52          type      coded;
53          libs      (utilityFunctionObjects);
54
55          codeEnd
56          #{
57              // Lookup U
58              Info<< "Looking up field U\n" << endl;
59              const auto& U = mesh().lookupObject<volVectorField>("U");
60
61              Info<< "Reading inlet velocity uInfx\n" << endl;
62
63              scalar ULeft = 0.0;
64              label leftI = mesh().boundaryMesh().findPatchID("left");
65              const auto& fvp = U.boundaryField()[leftI];
66              if (fvp.size())
67              {
68                  ULeft = fvp[0].x();
69              }
70              reduce(ULeft, maxOp<scalar>());
71
72              dimensionedScalar uInfx("uInfx", dimVelocity, ULeft);
73
74              Info<< "U at inlet = " << uInfx.value() << " m/s" << endl;
75
76
77              scalar magCylinder = 0.0;
78              label cylI = mesh().boundaryMesh().findPatchID("cylinder");
79              const auto& cylFvp = mesh().C().boundaryField()[cylI];
80              if (cylFvp.size())
81              {
82                  magCylinder = mag(cylFvp[0]);
83              }
84              reduce(magCylinder, maxOp<scalar>());
85
86              dimensionedScalar radius("radius", dimLength, magCylinder);
87
88              Info<< "Cylinder radius = " << radius.value() << " m" << endl;
89
90              volVectorField UA
91              (
92                  IOobject
93                  (
94                      "UA",
95                      mesh().time().timeName(),
96                      U.mesh(),
97                      IOobject::NO_READ,
98                      IOobject::AUTO_WRITE
99                  ),
100                  U
101              );
102
103              Info<< "\nEvaluating analytical solution" << endl;
104

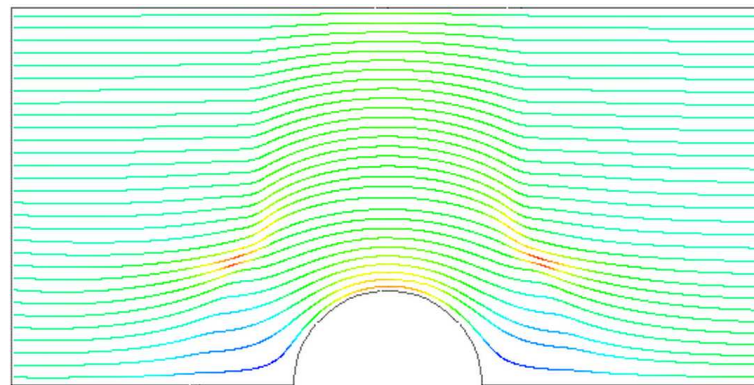
```

```

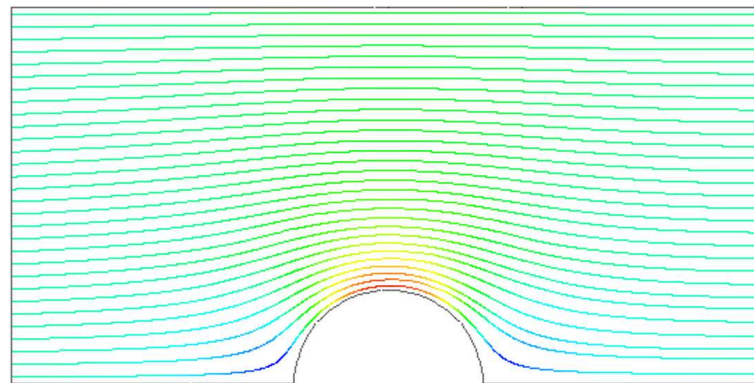
105         const volVectorField& centres = UA.mesh().C();
106         volScalarField magCentres(mag(centres));
107         volScalarField theta(acos((centres & vector(1,0,0))/magCentres));
108
109         volVectorField cs2theta
110         (
111             cos(2*theta)*vector(1,0,0)
112             + sin(2*theta)*vector(0,1,0)
113         );
114
115         UA = uInfX*(dimensionedVector(vector(1,0,0))
116             - pow((radius/magCentres),2)*cs2theta);
117
118         // Force writing of UA (since time has not changed)
119         UA.write();
120
121         volScalarField error("error", mag(U-UA)/mag(UA));
122
123         Info<<"Writing relative error in U to " << error.objectPath()
124             << endl;
125
126         error.write();
127     #};
128 }
129 }
130
131
132 // *****

```

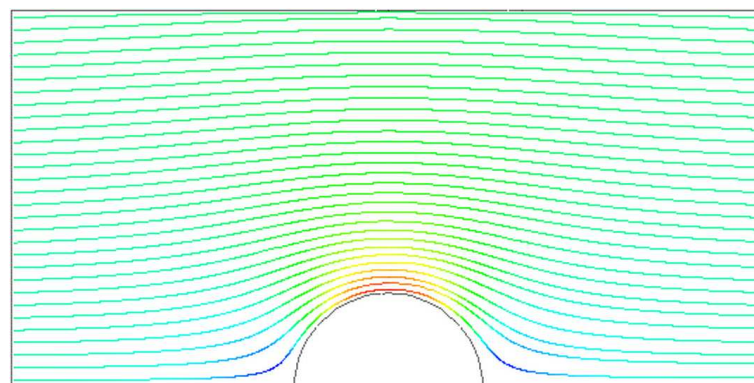
**potentialFoam** executes an iterative loop around the pressure equation which it solves in order that explicit terms relating to non-orthogonal correction in the Laplacian term may be updated in successive iterations. The number of iterations around the pressure equation is controlled by the **nNonOrthogonalCorrectors** keyword in the *fvSolution* dictionary. In the first instance we can set **nNonOrthogonalCorrectors** to 0 so that no loops are performed, *i.e.* the pressure equation is solved once, and there is no non-orthogonal correction. The solution is shown in Figure 2.17(a) (at  $t = 1$ , when the steady-state simulation is complete). We expect the solution to show smooth streamlines passing across the domain as in the analytical solution in Figure 2.17(c), yet there is clearly some error in the regions where there is high non-orthogonality in the mesh, *e.g.* at the join of blocks 0, 1 and 3. The case can be run a second time with some non-orthogonal correction by setting **nNonOrthogonalCorrectors** to 3. The solution shows smooth streamlines with no significant error due to non-orthogonality as shown in Figure 2.17(b).



(a) With no non-orthogonal correction



(b) With non-orthogonal correction



(c) Analytical solution

Figure 2.17: Streamlines of potential flow

## 2.3 Magnetohydrodynamic flow of a liquid

Tutorial path:

- [\\$FOAM\\_TUTORIALS/electromagnetics/mhdFoam/hartmann](#)

In this example we shall investigate the flow of an electrically-conducting liquid through a magnetic field. The problem belongs to the branch of fluid dynamics known as magnetohydrodynamics (MHD), simulated using the `mhdFoam` solver.

### 2.3.1 Problem specification

This case is known as the Hartmann problem, chosen as it contains an analytical solution with which `mhdFoam` can be validated. It is defined as follows:

**Solution domain** The domain is 2 dimensional and consists of flow along two parallel plates as shown in Fig. 2.18.

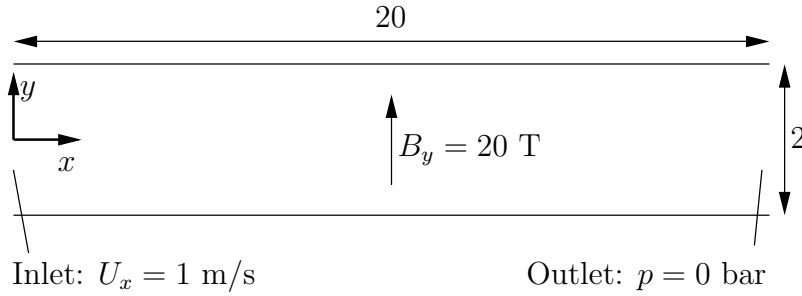


Figure 2.18: Geometry of the Hartmann problem

#### Governing equations

- Mass continuity for incompressible fluid

$$\nabla \cdot \mathbf{U} = 0 \quad (2.16)$$

- Momentum equation for incompressible fluid

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) + \nabla \cdot (2\mathbf{B}\Gamma_{\mathbf{B}\mathbf{U}}\mathbf{B}) + \nabla \cdot (\nu\mathbf{U}) + \nabla (\Gamma_{\mathbf{B}\mathbf{U}}\mathbf{B} \cdot \mathbf{B}) = -\nabla p \quad (2.17)$$

where  $\mathbf{B}$  is the magnetic flux density,  $\Gamma_{\mathbf{B}\mathbf{U}} = (2\mu\rho)^{-1}$ .

- Maxwell's equations

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \quad (2.18)$$

where  $\mathbf{E}$  is the electric field strength.

$$\nabla \cdot \mathbf{B} = 0 \quad (2.19)$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t} = \mathbf{J} \quad (2.20)$$

assuming  $\partial \mathbf{D} / \partial t \ll \mathbf{J}$ . Here,  $\mathbf{H}$  is the magnetic field strength,  $\mathbf{J}$  is the current density and  $\mathbf{D}$  is the electric flux density.

- Charge continuity

$$\nabla \cdot \mathbf{J} = 0 \quad (2.21)$$

- Constitutive law

$$\mathbf{B} = \mu \mathbf{H} \quad (2.22)$$

- Ohm's law

$$\mathbf{J} = \sigma (\mathbf{E} + \mathbf{U} \times \mathbf{B}) \quad (2.23)$$

- Combining Equation 2.18, Equation 2.20, Equation 2.23, and taking the curl

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \cdot (\mathbf{U} \mathbf{B}) - \nabla \cdot (\phi_{\mathbf{B}} \mathbf{U}) - \nabla \cdot (\Gamma_{\mathbf{B}} \mathbf{B}) = 0 \quad (2.24)$$

### Boundary conditions

- `inlet` is specified the `inlet` condition with fixed velocity  $\mathbf{U} = (1, 0, 0)$  m/s;
- `outlet` is specified as the `outlet` with fixed pressure  $p = 0$  Pa;
- `upperWall` is specified as a `wall` where  $\mathbf{B} = (0, 20, 0)$  T.
- `lowerWall` is specified as a `wall` where  $\mathbf{B} = (0, 20, 0)$  T.
- `front` and `back` boundaries are specified as `empty`.

**Initial conditions**  $\mathbf{U} = \mathbf{0}$  m/s,  $p = 100$  Pa,  $\mathbf{B} = (0, 20, 0)$  T.

### Transport properties

- Kinematic viscosity  $\nu = 1$  Pa s
- Density  $\rho = 1$  kg m/s
- Electrical conductivity  $\sigma = 1$  ( $\Omega$  m)<sup>-1</sup>
- Permeability  $\mu = 1$  H/m

**Solver name** `mhdFoam`: an incompressible laminar magneto-hydrodynamics code.

**Case name** `hartmann` case located in the `$FOAM_TUTORIALS/electromagnetics/mhd-Foam` directory.

## 2.3.2 Mesh generation

The geometry is simply modelled with 100 cells in the  $x$ -direction and 40 cells in the  $y$ -direction; the set of vertices and blocks are given in the mesh description file below:

```

1  /*-----* C++ *-----*/
2  |=====|
3  | \ \ /  | F ield      | OpenFOAM: The Open Source CFD Toolbox
4  |  \ \   | O peration  | Version: v2506
5  |   \ \  | A nd        | Website: www.openfoam.com
6  |    \ \  | M anipulation|
7  |-----*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        dictionary;
13     object       blockMeshDict;
14 }

```

```

15 // * * * * *
16
17 scale 1;
18
19 vertices
20 (
21     (0 -1 0)
22     (20 -1 0)
23     (20 1 0)
24     (0 1 0)
25     (0 -1 0.1)
26     (20 -1 0.1)
27     (20 1 0.1)
28     (0 1 0.1)
29 );
30
31 blocks
32 (
33     hex (0 1 2 3 4 5 6 7) (100 40 1) simpleGrading (1 1 1)
34 );
35
36 boundary
37 (
38     inlet
39     {
40         type patch;
41         faces
42         (
43             (0 4 7 3)
44         );
45     }
46     outlet
47     {
48         type patch;
49         faces
50         (
51             (2 6 5 1)
52         );
53     }
54     lowerWall
55     {
56         type patch;
57         faces
58         (
59             (1 5 4 0)
60         );
61     }
62     upperWall
63     {
64         type patch;
65         faces
66         (
67             (3 7 6 2)
68         );
69     }
70     frontAndBack
71     {
72         type empty;
73         faces
74         (
75             (0 3 2 1)
76             (4 5 6 7)
77         );
78     }
79 );
80
81 // *****
82

```

### 2.3.3 Running the case

The user can run the case and view results in ParaView. It is also useful at this stage to run the Ucomponents utility to convert the  $\mathbf{U}$  vector field into individual scalar components. MHD flow is governed by, amongst other things, the Hartmann number which is a measure of the ratio of electromagnetic body force to viscous force

$$M = BL\sqrt{\frac{\sigma}{\rho\nu}} \quad (2.25)$$

where  $L$  is the characteristic length scale. In this case with  $B_y = 20$  T,  $M = 20$  and the electromagnetic body forces dominate the viscous forces. Consequently with the flow fairly steady at  $t = 2$  s the velocity profile is almost planar, viewed at a cross section midway along the domain  $x = 10$  m. The user can plot a graph of the profile of  $U_x$  in `dxFoam`. Now the user should reduce the magnetic flux density  $\mathbf{B}$  to 1 T and re-run the

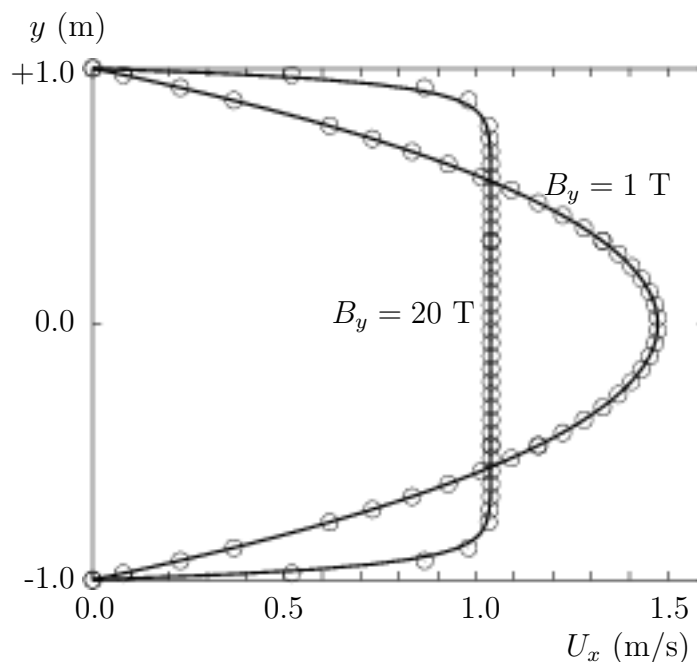


Figure 2.19: Velocity profile in the Hartmann problem for  $B_y = 1$  T and  $B_y = 20$  T.

code and `Ucomponents`. In this case,  $M = 1$  and the electromagnetic body forces no longer dominate. The velocity profile consequently takes on the parabolic form, characteristic of Poiseuille flow as shown in Figure 2.19. To validate the code the analytical solution for the velocity profile  $U_x$  is superimposed in Figure 2.19, given by:

$$\frac{U_x(y)}{U_x(0)} = \frac{\cosh M - \cosh M(y/L)}{\cosh M - 1} \quad (2.26)$$

where the characteristic length  $L$  is half the width of the domain, *i.e.* 1 m.



# Chapter 3

## Compressible flow

## 3.1 Steady turbulent flow over a backward-facing step

Tutorial path:

- [\\$FOAM\\_TUTORIALS/incompressible/simpleFoam/pitzDaily](#)

In this example we shall investigate steady turbulent flow over a backward-facing step. The problem description is taken from one used by Pitz and Daily in an experimental investigation [\*\*] against which the computed solution can be compared. This example introduces the following OpenFOAM features for the first time:

- generation of a mesh using `blockMesh` using full mesh grading capability;
- steady turbulent flow.

### 3.1.1 Problem specification

The problem is defined as follows:

**Solution domain** The domain is 2 dimensional, consisting of a short inlet, a backward-facing step and converging nozzle at outlet as shown in Figure 3.1.

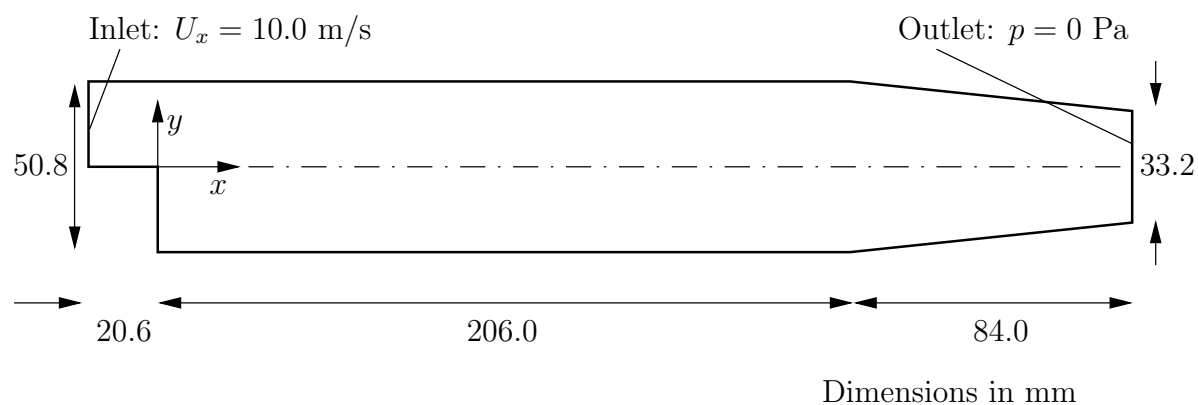


Figure 3.1: Geometry of backward-facing step

#### Governing equations

- Mass continuity for incompressible flow

$$\nabla \cdot \mathbf{U} = 0 \quad (3.1)$$

- Steady flow momentum equation

$$\nabla \cdot (\mathbf{U}\mathbf{U}) + \nabla \cdot \mathbf{R} = -\nabla p \quad (3.2)$$

where  $p$  is kinematic pressure and (in slightly over-simplistic terms)  $\mathbf{R} = \nu_{eff} \nabla \mathbf{U}$  is the viscous stress term with an effective kinematic viscosity  $\nu_{eff}$ , calculated from selected transport and turbulence models.

**Initial conditions**  $U = 0$  m/s,  $p = 0$  Pa — required in OpenFOAM input files but not necessary for the solution since the problem is steady-state.

#### Boundary conditions

- Inlet (left) with fixed velocity  $\mathbf{U} = (10, 0, 0)$  m/s;
- Outlet (right) with fixed pressure  $p = 0$  Pa;
- No-slip walls on other boundaries.

### Transport properties

- Kinematic viscosity of air  $\nu = \mu/\rho = 18.1 \times 10^{-6}/1.293 = 14.0 \text{ } \mu\text{m}^2/\text{s}$

### Turbulence model

- Standard  $k - \epsilon$ ;
- Coefficients:  $C_\mu = 0.09$ ;  $C_1 = 1.44$ ;  $C_2 = 1.92$ ;  $\alpha_k = 1$ ;  $\alpha_\epsilon = 0.76923$ .

**Solver name** `simpleFoam`: an implementation for steady incompressible flow.

**Case name** `pitzDaily`, located in the `$FOAM_TUTORIALS/incompressible/simpleFoam` directory.

The problem is solved using `simpleFoam`, so-called as it is an implementation for steady flow using the SIMPLE algorithm. The solver has full access to all the turbulence models in the `incompressibleTurbulenceModels` library and the non-Newtonian models `incompressibleTransportModels` library of the standard OpenFOAM release.

### 3.1.2 Mesh generation

We expect that the flow in this problem is reasonably complex and an optimum solution will require grading of the mesh. In general, the regions of highest shear are particularly critical, requiring a finer mesh than in the regions of low shear. We can anticipate where high shear will occur by considering what the solution might be in advance of any calculation. At the inlet we have strong uniform flow in the  $x$  direction and, as it passes over the step, it generates shear on the fluid below, generating a vortex in the bottom half of the domain. The regions of high shear will therefore be close to the centreline of the domain and close to the walls.

The domain is subdivided into 12 blocks as shown in Figure 3.2.

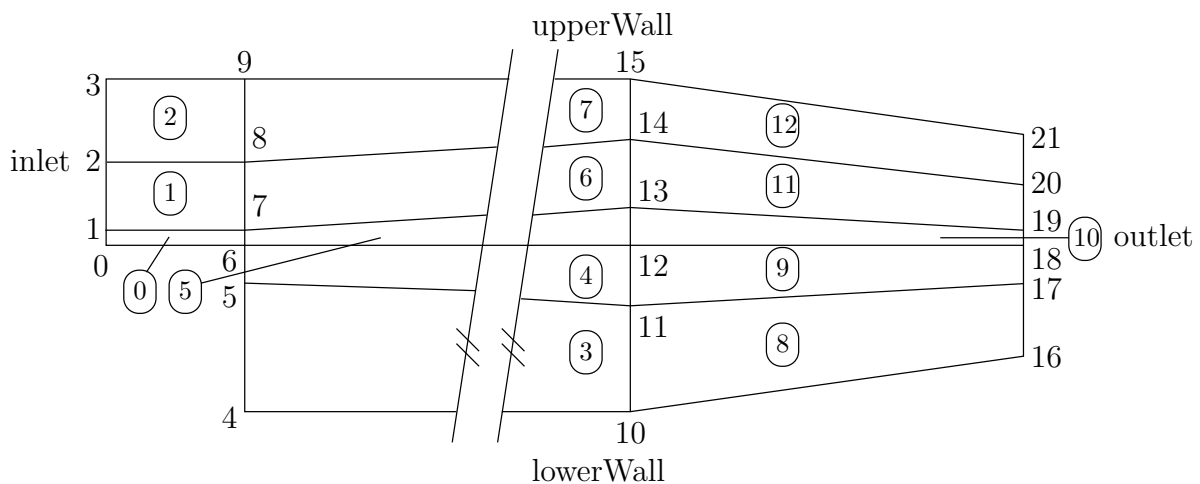


Figure 3.2: Blocks in backward-facing step

The mesh is 3 dimensional, as always in OpenFOAM, so in Figure 3.2 we are viewing the back plane along  $z = -0.5$ . The full set of vertices and blocks are given in the mesh description file below:

```

1  /*----- C++ -----*/
2  |=====|
3  | \ \ \ \ | F i e l d | OpenFOAM: The Open Source CFD Toolbox
4  | \ \ \ \ | O p e r a t i o n | Version: v2506
5  | \ \ \ \ | A n d | Website: www.openfoam.com
6  | \ \ \ \ | M a n i p u l a t i o n |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class          dictionary;
13     object         blockMeshDict;
14 }
15 // *****
16
17 scale 0.001;
18
19 vertices
20 (
21     (-20.6 0 -0.5)
22     (-20.6 25.4 -0.5)
23     (0 -25.4 -0.5)
24     (0 0 -0.5)
25     (0 25.4 -0.5)
26     (206 -25.4 -0.5)
27     (206 0 -0.5)
28     (206 25.4 -0.5)
29     (290 -16.6 -0.5)
30     (290 0 -0.5)
31     (290 16.6 -0.5)
32
33     (-20.6 0 0.5)
34     (-20.6 25.4 0.5)
35     (0 -25.4 0.5)
36     (0 0 0.5)
37     (0 25.4 0.5)
38     (206 -25.4 0.5)
39     (206 0 0.5)
40     (206 25.4 0.5)
41     (290 -16.6 0.5)
42     (290 0 0.5)
43     (290 16.6 0.5)
44 );
45
46 negY
47 (
48     (2 4 1)
49     (1 3 0.3)
50 );
51
52 posY
53 (
54     (1 4 2)
55     (2 3 4)
56     (2 4 0.25)
57 );
58
59 posYR
60 (
61     (2 1 1)
62     (1 1 0.25)
63 );
64
65
66 blocks
67 (
68     hex (0 3 4 1 11 14 15 12)
69     (18 30 1)
70     simpleGrading (0.5 $posY 1)
71
72     hex (2 5 6 3 13 16 17 14)
73     (180 27 1)
74     edgeGrading (4 4 4 4 $negY 1 1 $negY 1 1 1 1)
75
76     hex (3 6 7 4 14 17 18 15)
77     (180 30 1)
78     edgeGrading (4 4 4 4 $posY $posYR $posYR $posY 1 1 1 1)
79
80     hex (5 8 9 6 16 19 20 17)
81     (25 27 1)
82     simpleGrading (2.5 1 1)
83
84     hex (6 9 10 7 17 20 21 18)
85     (25 30 1)

```

```

86     simpleGrading (2.5 $posYR 1)
87 );
88
89 edges
90 (
91 );
92
93 boundary
94 (
95     inlet
96     {
97         type patch;
98         faces
99         (
100             (0 1 12 11)
101         );
102     }
103     outlet
104     {
105         type patch;
106         faces
107         (
108             (8 9 20 19)
109             (9 10 21 20)
110         );
111     }
112     upperWall
113     {
114         type wall;
115         faces
116         (
117             (1 4 15 12)
118             (4 7 18 15)
119             (7 10 21 18)
120         );
121     }
122     lowerWall
123     {
124         type wall;
125         faces
126         (
127             (0 3 14 11)
128             (3 2 13 14)
129             (2 5 16 13)
130             (5 8 19 16)
131         );
132     }
133     frontAndBack
134     {
135         type empty;
136         faces
137         (
138             (0 3 4 1)
139             (2 5 6 3)
140             (3 6 7 4)
141             (5 8 9 6)
142             (6 9 10 7)
143             (11 14 15 12)
144             (13 16 17 14)
145             (14 17 18 15)
146             (16 19 20 17)
147             (17 20 21 18)
148         );
149     }
150 );
151
152 // *****
153
```

A major feature of this problem is the use of the full mesh grading capability of **blockMesh** that is described in section 4.3.1 of the User Guide. The user can see that blocks 4,5 and 6 use the full list of 12 expansion ratios. The expansion ratios correspond to each edge of the block, the first 4 to the edges aligned in the local  $x_1$  direction, the second 4 to the edges in the local  $x_2$  direction and the last 4 to the edges in the local  $x_3$  direction. In blocks 4, 5, and 6, the ratios are equal for all edges in the local  $x_1$  and  $x_3$  directions but not for the edges in the  $x_2$  direction that corresponds in all blocks to the global  $y$ . If we consider the ratios used in relation to the block definition in section 4.3.1 of the User Guide, we realize that different gradings have been prescribed along the left and right edges in blocks 4,5 and 6 in Figure 3.2. The purpose of this differential grading

is to generate a fine mesh close to the most critical region of flow, the corner of the step, and allow it to expand into the rest of the domain.

The mesh can be generated using `blockMesh` from the command line and viewed as described in previous examples.

### 3.1.3 Boundary conditions and initial fields

Edit the case files to set the initial and boundary fields for velocity  $\mathbf{U}$ , pressure  $p$ , turbulent kinetic energy  $k$  and dissipation rate  $\varepsilon$ . The boundary conditions can be specified as: the upper and lower walls are set to `Wall`, the left patch to `Inlet` and the right patch to `Outlet`. These physical boundary conditions require us to specify a `fixedValue` at the inlet on  $\mathbf{U}$ ,  $k$  and  $\varepsilon$ .  $\mathbf{U}$  is given in the problem specification, but the values of  $k$  and  $\varepsilon$  must be chosen by the user in a similar manner to that described in section 2.1.8.1 of the User Guide. We assume that the inlet turbulence is isotropic and estimate the fluctuations to be 5% of  $\mathbf{U}$  at the inlet. We have

$$U'_x = U'_y = U'_z = \frac{5}{100} 10 = 0.5 \text{ m/s} \quad (3.3)$$

and

$$k = \frac{3}{2} (0.5)^2 = 0.375 \text{ m}^2/\text{s}^2 \quad (3.4)$$

If we estimate the turbulent length scale  $l$  to be 10% of the width of the inlet then

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{l} = \frac{0.09^{0.75} 0.375^{1.5}}{0.1 \times 25.4 \times 10^{-3}} = 14.855 \text{ m}^2/\text{s}^3 \quad (3.5)$$

At the outlet we need only specify the pressure  $p = 0\text{Pa}$ .

### 3.1.4 Case control

The choices of *fvSchemes* are as follows: the `timeScheme` should be `steadyState`; the `gradSchemes` and `laplacianSchemes` should be set as default to `Gauss`; and, the `divSchemes` should be set to `upwind` to ensure boundedness.

Special attention should be paid to the solver settings of the *fvSolution* dictionary. Although the top level `simpleFoam` code contains only equations for  $p$  and  $\mathbf{U}$ , the turbulence model solves equations for  $k$ ,  $\varepsilon$  and  $\mathbf{R}$ , and tolerance settings are required for all 5 equations. A `tolerance` of  $10^{-5}$  and `relTol` of 0.1 are acceptable for all variables with the exception of  $p$  when  $10^{-6}$  and 0.01 are recommended. Under-relaxation of the solution is required since the problem is steady. A `relaxationFactor` of 0.7 is acceptable for  $\mathbf{U}$ ,  $k$ , and  $\varepsilon$  but 0.3 is required for  $p$  to avoid numerical instability.

Finally, in the *controlDict* dictionary, the time step `deltaT` should be set to 1 since in steady state cases such as this is effectively an iteration counter. With benefit of hindsight we know that the solution requires 1000 iterations reach reasonable convergence, hence `endTime` is set to 1000. Ensure that the `writeInterval` is sufficiently high, *e.g.* 50, that you will not fill the hard disk with data during run time.

### 3.1.5 Running the case and post-processing

Run the case and post-process the results. After a few iterations, *e.g.* 50, a vortex develops beneath the corner of the step that is the height of the step but narrow in the  $x$ -direction as shown by the vector plot of velocities is shown Figure 3.3(a). Over several iterations

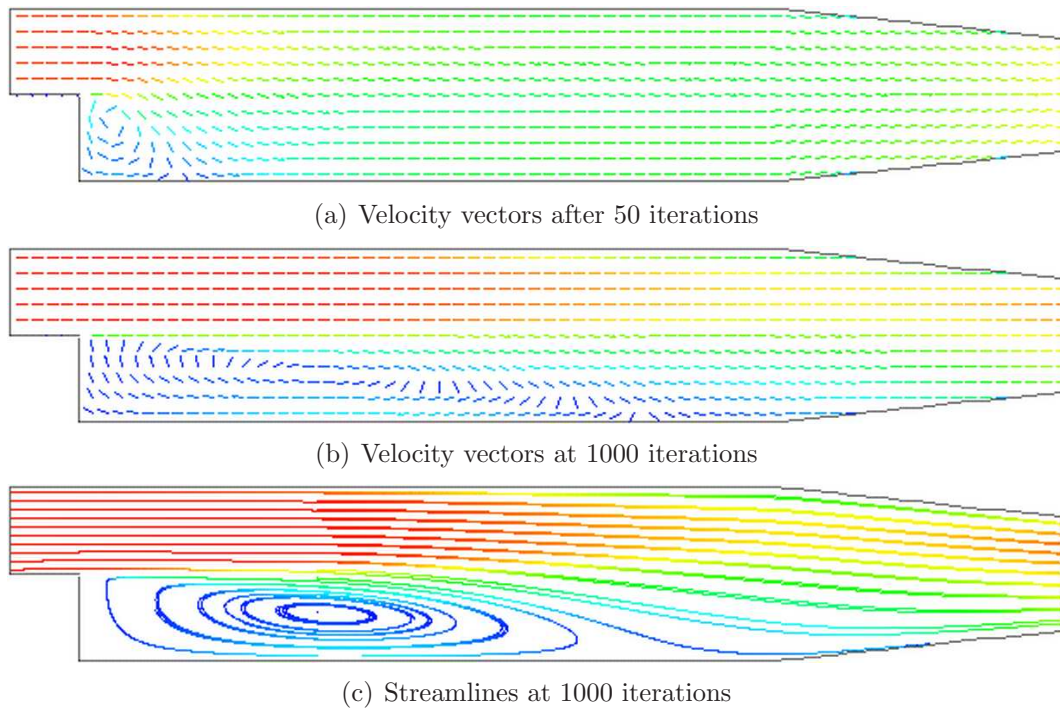


Figure 3.3: Development of a vortex in the backward-facing step.

the vortex stretches in the  $x$ -direction from the step to the outlet until at 1000 iterations the system reaches a steady-state in which the vortex is fully developed as shown in Figure 3.3(b-c).

## 3.2 Supersonic flow over a forward-facing step

Tutorial path:

- [\\$FOAM\\_TUTORIALS/compressible/sonicFoam/laminar/forwardStep](#)

In this example we shall investigate supersonic flow over a forward-facing step. The problem description involves a flow of Mach 3 at an inlet to a rectangular geometry with a step near the inlet region that generates shock waves.

This example introduces the following OpenFOAM features for the first time:

- supersonic flow;

### 3.2.1 Problem specification

The problem is defined as follows:

**Solution domain** The domain is 2 dimensional and consists of a short inlet section followed by a forward-facing step of 20% the height of the section as shown in Figure 3.4

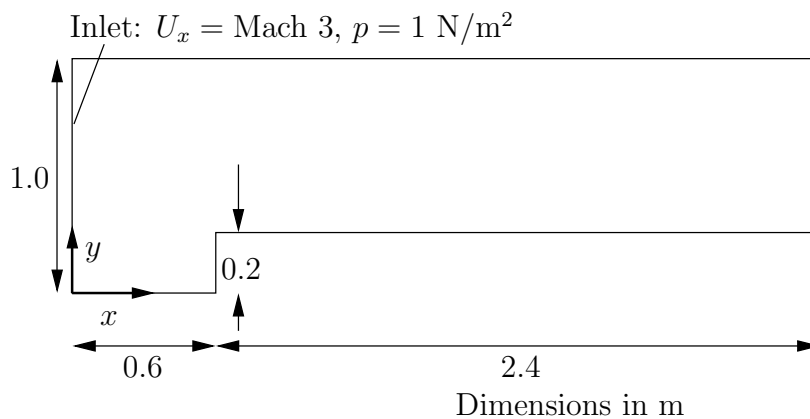


Figure 3.4: Geometry of the forward step geometry

#### Governing equations

- Mass continuity

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \quad (3.6)$$

- Ideal gas

$$p = \rho R T \quad (3.7)$$

- Momentum equation for Newtonian fluid

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot (\rho \mathbf{U} \mathbf{U}) - \nabla \cdot (\mu \nabla \mathbf{U}) = -\nabla p \quad (3.8)$$

- Energy equation for fluid (ignoring some viscous terms),  $e = C_v T$ , with Fourier's Law  $\mathbf{q} = -k \nabla T$

$$\frac{\partial \rho e}{\partial t} + \nabla \cdot (\rho \mathbf{U} e) - \nabla \cdot \left( \frac{k}{C_v} \right) \nabla e = p \nabla \cdot \mathbf{U} \quad (3.9)$$



**Initial conditions**  $U = 0$  m/s,  $p = 1$  Pa,  $T = 1$  K.

### Boundary conditions

- Inlet (left) with `fixedValue` for velocity  $U = 3$  m/s = Mach 3, pressure  $p = 1$  Pa and temperature  $T = 1$  K;
- Outlet (right) with `zeroGradient` on  $U$ ,  $p$  and  $T$ ;
- No-slip adiabatic wall (bottom);
- Symmetry plane (top).

### Transport properties

- Dynamic viscosity of air  $\mu = 18.1 \mu\text{Pa s}$

### Thermodynamic properties

- Specific heat at constant volume  $C_v = 1.78571$  J/kg K
- Gas constant  $R = 0.714286$  J/kg K
- Conductivity  $k = 32.3 \mu\text{W/m K}$

**Case name** `forwardStep` case located in the `$FOAM_TUTORIALS/compressible/sonicFoam/laminar` directory.

**Solver name** `sonicFoam`: an implementation for compressible trans-sonic/supersonic laminar gas flow.

The case is designed such that the speed of sound of the gas  $c = \sqrt{\gamma RT} = 1$  m/s, the consequence being that the velocities are directly equivalent to the Mach number, *e.g.* the inlet velocity of 3 m/s is equivalent to Mach 3. This speed of sound calculation can be verified using the relationship for a perfect gas,  $C_p - C_v = R$ , *i.e.* the ratio of specific heats

$$\gamma = C_p/C_v = \frac{R}{C_v} + 1 \quad (3.10)$$

## 3.2.2 Mesh generation

The mesh used in this case is relatively simple, specified with uniform rectangular cells of length 0.06 m in the  $x$  direction and 0.05 m in the  $y$  direction. The geometry can simply be divided into 3 blocks, one below the top of the step, and two above the step, one either side of the step front. The full set of vertices and blocks are given in the mesh description file below:

```

1  /*-----*-- C++ --*-----*\
2  |=====|
3  |  \  /  | F ield          | OpenFOAM: The Open Source CFD Toolbox
4  |   \ /   | O peration     | Version: v2506
5  |    \/    | A nd           | Website: www.openfoam.com
6  |     \/    | M anipulation  |
7  \*-----*--*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class          dictionary;
13     object         blockMeshDict;
14  }
15  // *****
16
17  scale 1;
```

```

18
19 vertices
20 (
21     (0 0 -0.05)
22     (0.6 0 -0.05)
23     (0 0.2 -0.05)
24     (0.6 0.2 -0.05)
25     (3 0.2 -0.05)
26     (0 1 -0.05)
27     (0.6 1 -0.05)
28     (3 1 -0.05)
29     (0 0 0.05)
30     (0.6 0 0.05)
31     (0 0.2 0.05)
32     (0.6 0.2 0.05)
33     (3 0.2 0.05)
34     (0 1 0.05)
35     (0.6 1 0.05)
36     (3 1 0.05)
37 );
38
39 blocks
40 (
41     hex (0 1 3 2 8 9 11 10) (25 10 1) simpleGrading (1 1 1)
42     hex (2 3 6 5 10 11 14 13) (25 40 1) simpleGrading (1 1 1)
43     hex (3 4 7 6 11 12 15 14) (100 40 1) simpleGrading (1 1 1)
44 );
45
46 edges
47 (
48 );
49
50 boundary
51 (
52     inlet
53     {
54         type patch;
55         faces
56         (
57             (0 8 10 2)
58             (2 10 13 5)
59         );
60     }
61     outlet
62     {
63         type patch;
64         faces
65         (
66             (4 7 15 12)
67         );
68     }
69     bottom
70     {
71         type symmetryPlane;
72         faces
73         (
74             (0 1 9 8)
75         );
76     }
77     top
78     {
79         type symmetryPlane;
80         faces
81         (
82             (5 13 14 6)
83             (6 14 15 7)
84         );
85     }
86     obstacle
87     {
88         type patch;
89         faces
90         (
91             (1 3 11 9)
92             (3 4 12 11)
93         );
94     }
95 );
96
97 mergePatchPairs
98 (
99 );
100
101
102 // *****

```

### 3.2.3 Running the case

The case approaches a steady-state at some time after 10 s. The results for pressure at 2 s are shown in Figure 3.5. The results clearly show discontinuities in pressure, *i.e.* shock waves, emanating from ahead of the base of the step.

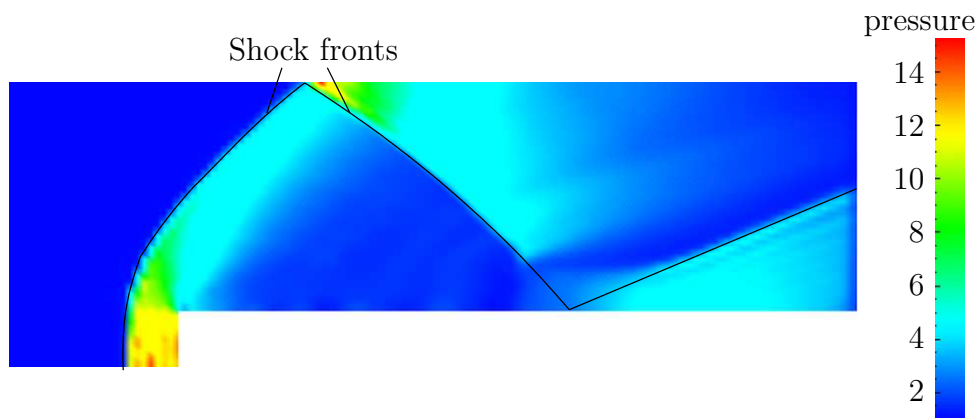


Figure 3.5: Shock fronts in the forward step problem

### 3.2.4 Exercise

The user can examine the effect on the solution of increasing the inlet velocity.

### 3.3 Decompression of a tank internally pressurised with water

Tutorial path:

- [\\$FOAM\\_TUTORIALS/compressible/sonicLiquidFoam/decompressionTank](#)

In this example we shall investigate a problem of rapid opening of a pipe valve close to a pressurised liquid-filled tank. The prominent feature of the result in such cases is the propagation of pressure waves which must therefore be modelled as a compressible liquid.

This tutorial introduces the following OpenFOAM features for the first time:

- Mesh refinement
- Pressure waves in liquids

#### 3.3.1 Problem specification

**Solution domain** The domain is 2 dimensional and consists of a tank with a small outflow pipe as shown in Figure 3.6

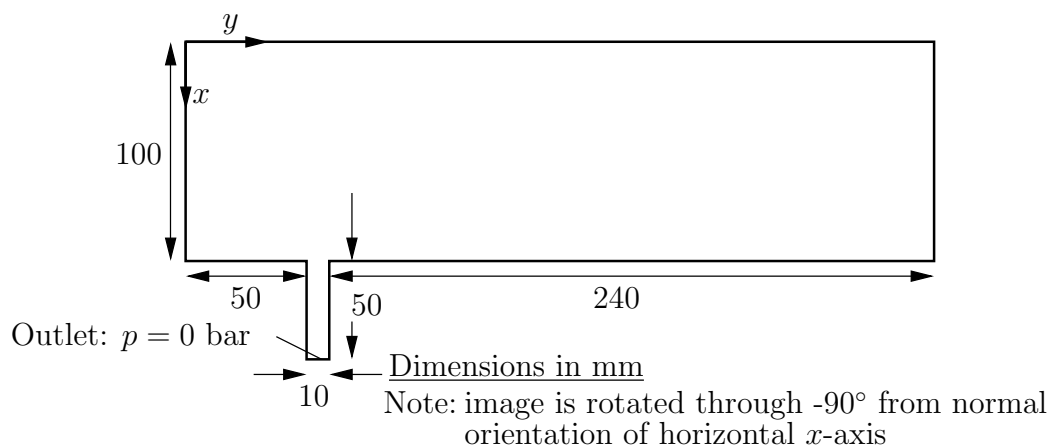


Figure 3.6: Geometry of a tank with outflow pipe

**Governing equations** This problem requires a model for compressibility  $\psi$  in the fluid in order to be able to resolve waves propagating at a finite speed. A barotropic relationship is used to relate density  $\rho$  and pressure  $p$  are related to  $\psi$ .

- Mass continuity

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \quad (3.11)$$

- The barotropic relationship

$$\frac{\partial \rho}{\partial p} = \frac{\rho}{K} = \psi \quad (3.12)$$

where  $K$  is the bulk modulus

- Equation 3.12 is linearised as

$$\rho \approx \rho_0 + \psi (p - p_0) \quad (3.13)$$

where  $\rho_0$  and  $p_0$  are the reference density and pressure respectively such that  $\rho(p_0) = \rho_0$ .

- Momentum equation for Newtonian fluid

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot (\rho \mathbf{U} \mathbf{U}) - \nabla \cdot \mu \nabla \mathbf{U} = -\nabla p \quad (3.14)$$

### *Boundary conditions*

- `outerWall` is specified the `wall` condition;
- `axis` is specified as the `symmetryPlane`;
- `nozzle` is specified as a `pressureOutlet` where  $p = 0$  bar.
- `front` and `back` boundaries are specified as `empty`.

*Initial conditions*  $\mathbf{U} = \mathbf{0}$  m/s,  $p = 100$  bar.

### *Transport properties*

- Dynamic viscosity of water  $\mu = 1.0$  mPa s

### *Thermodynamic properties*

- Density of water  $\rho = 1000$  kg/m<sup>3</sup>
- Reference pressure  $p_0 = 1$  bar
- Compressibility of water  $\psi = 4.54 \times 10^{-7}$  s<sup>2</sup>/m<sup>2</sup>

*Solver name* `sonicLiquidFoam`: a solver for compressible sonic laminar liquid flow.

*Case name* `decompressionTank` case located in the `$FOAM_TUTORIALS/compressible/-sonicLiquidFoam` directory.

## 3.3.2 Mesh Generation

The full geometry is modelled in this case; the set of vertices and blocks are given in the mesh description file below:

```

1  /*-----*----- C++ *-----*/
2  |=====|
3  | \ \ / F i e l d | OpenFOAM: The Open Source CFD Toolbox |
4  | \ \ / O p e r a t i o n | Version: v2506 |
5  | \ \ / A n d | Website: www.openfoam.com |
6  | \ \ / M a n i p u l a t i o n | |
7  /*-----*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        dictionary;
13     object       blockMeshDict;
14 }
15 // *****
16
17 scale 0.1;
18
19 vertices
20 (
21     (0 0 -0.1)
22     (1 0 -0.1)
23     (0 0.5 -0.1)
24     (1 0.5 -0.1)
25     (1.5 0.5 -0.1)
26     (0 0.6 -0.1)
27     (1 0.6 -0.1)
28     (1.5 0.6 -0.1)
29     (0 3 -0.1)
30     (1 3 -0.1)
31     (0 0 0.1)

```

```

32     (1 0 0.1)
33     (0 0.5 0.1)
34     (1 0.5 0.1)
35     (1.5 0.5 0.1)
36     (0 0.6 0.1)
37     (1 0.6 0.1)
38     (1.5 0.6 0.1)
39     (0 3 0.1)
40     (1 3 0.1)
41 );
42
43 blocks
44 (
45     hex (0 1 3 2 10 11 13 12) (30 20 1) simpleGrading (1 1 1)
46     hex (2 3 6 5 12 13 16 15) (30 5 1) simpleGrading (1 1 1)
47     hex (3 4 7 6 13 14 17 16) (25 5 1) simpleGrading (1 1 1)
48     hex (5 6 9 8 15 16 19 18) (30 95 1) simpleGrading (1 1 1)
49 );
50
51 edges
52 (
53 );
54
55 boundary
56 (
57     outerWall
58     {
59         type wall;
60         faces
61         (
62             (0 1 11 10)
63             (1 3 13 11)
64             (3 4 14 13)
65             (7 6 16 17)
66             (6 9 19 16)
67             (9 8 18 19)
68         );
69     }
70     axis
71     {
72         type symmetryPlane;
73         faces
74         (
75             (0 10 12 2)
76             (2 12 15 5)
77             (5 15 18 8)
78         );
79     }
80     nozzle
81     {
82         type patch;
83         faces
84         (
85             (4 7 17 14)
86         );
87     }
88     back
89     {
90         type empty;
91         faces
92         (
93             (0 2 3 1)
94             (2 5 6 3)
95             (3 6 7 4)
96             (5 8 9 6)
97         );
98     }
99     front
100    {
101        type empty;
102        faces
103        (
104            (10 11 13 12)
105            (12 13 16 15)
106            (13 14 17 16)
107            (15 16 19 18)
108        );
109    }
110 );
111
112 mergePatchPairs
113 (
114 );
115
116

```

117 // \*\*\*\*\* //

In order to improve the numerical accuracy, we shall use the reference level of 1 bar for the pressure field. Note that both the internal field level and the boundary conditions are offset by the reference level.

### 3.3.3 Preparing the Run

Before we commence the setup of the calculation, we need to consider the characteristic velocity of the phenomenon we are trying to capture. In the case under consideration, the fluid velocity will be very small, but the pressure wave will propagate with the speed of sound in water. The speed of sound is calculated as:

$$c = \sqrt{\frac{1}{\psi}} = \sqrt{\frac{1}{4.54 \times 10^{-7}}} = 1483.2 \text{ m/s.} \quad (3.15)$$

For the mesh described above, the characteristic mesh size is approximately 2 mm (note the scaling factor of 0.1 in the *blockMeshDict* file). Using

$$Co = \frac{U \Delta t}{\Delta x} \quad (3.16)$$

a reasonable time step is around  $\Delta t = 5 \times 10^{-7}$  s, giving the *Co* number of 0.35, based on the speed of sound. Also, note that the reported *Co* number by the code (associated with the convective velocity) will be two orders of magnitude smaller. As we are interested in the pressure wave propagation, we shall set the simulation time to 0.25 ms. For reference, the *controlDict* file is quoted below.

```

1  /*-----* C++ -*-----*\
2  |=====|
3  | \ \ \ / F ield      | OpenFOAM: The Open Source CFD Toolbox |
4  | \ \ \ / O peration   | Version: v2506                      |
5  | \ \ \ / A nd         | Website: www.openfoam.com           |
6  | \ \ \ / M anipulation |
7  /*-----*
8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        dictionary;
13     object       controlDict;
14 }
15 // *****
16
17 application      sonicLiquidFoam;
18
19 startFrom        startTime;
20
21 startTime        0;
22
23 stopAt           endTime;
24
25 endTime          0.0001;
26
27 deltaT           5e-07;
28
29 writeControl      timeStep;
30
31 writeInterval     20;
32
33 purgeWrite        0;
34
35 writeFormat       ascii;
36
37 writePrecision    6;
38
39 writeCompression off;
40
41 timeFormat        general;

```

```

42
43 timePrecision 6;
44
45 runtimeModifiable true;
46
47
48 // ***** //

```

### 3.3.4 Running the case

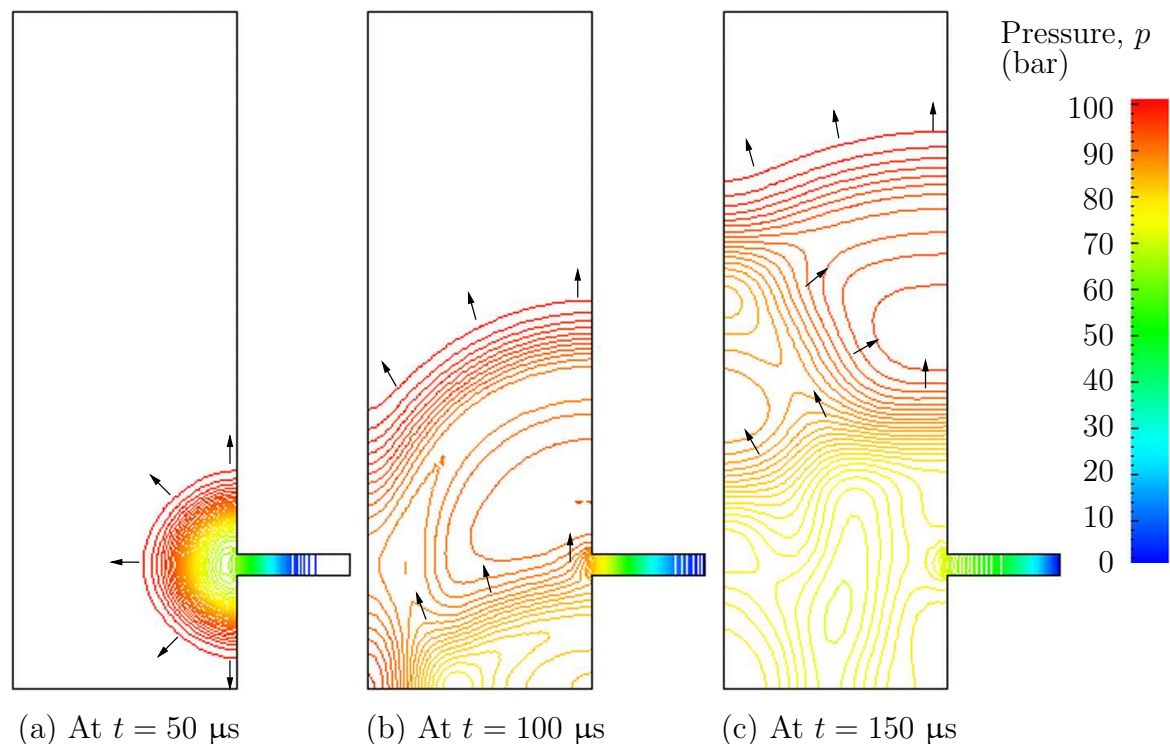


Figure 3.7: Propagation of pressure waves

The user can run the case and view results in `dxFoam`. The liquid flows out through the nozzle causing a wave to move along the nozzle. As it reaches the inlet to the tank, some of the wave is transmitted into the tank and some of it is reflected. While a wave is reflected up and down the inlet pipe, the waves transmitted into the tank expand and propagate through the tank. In Figure 3.7, the pressures are shown as contours so that the wave fronts are more clearly defined than if plotted as a normal isoline plot.

If the simulation is run for a long enough time for the reflected wave to return to the pipe, we can see that negative absolute pressure is detected. The modelling permits this and has some physical basis since liquids can support tension, *i.e.* negative pressures. In reality, however, impurities or dissolved gases in liquids act as sites for cavitation, or vapourisation/boiling, of the liquid due to the low pressure. Therefore in practical situations, we generally do not observe pressures falling below the vapourisation pressure of the liquid; not at least for longer than it takes for the cavitation process to occur.

### 3.3.5 Improving the solution by refining the mesh

Looking at the evolution of the resulting pressure field in time, we can clearly see the propagation of the pressure wave into the tank and numerous reflections from the inside walls. It is also obvious that the pressure wave is smeared over a number of cells. We shall now refine the mesh and reduce the time step to obtain a sharper front resolution. Simply



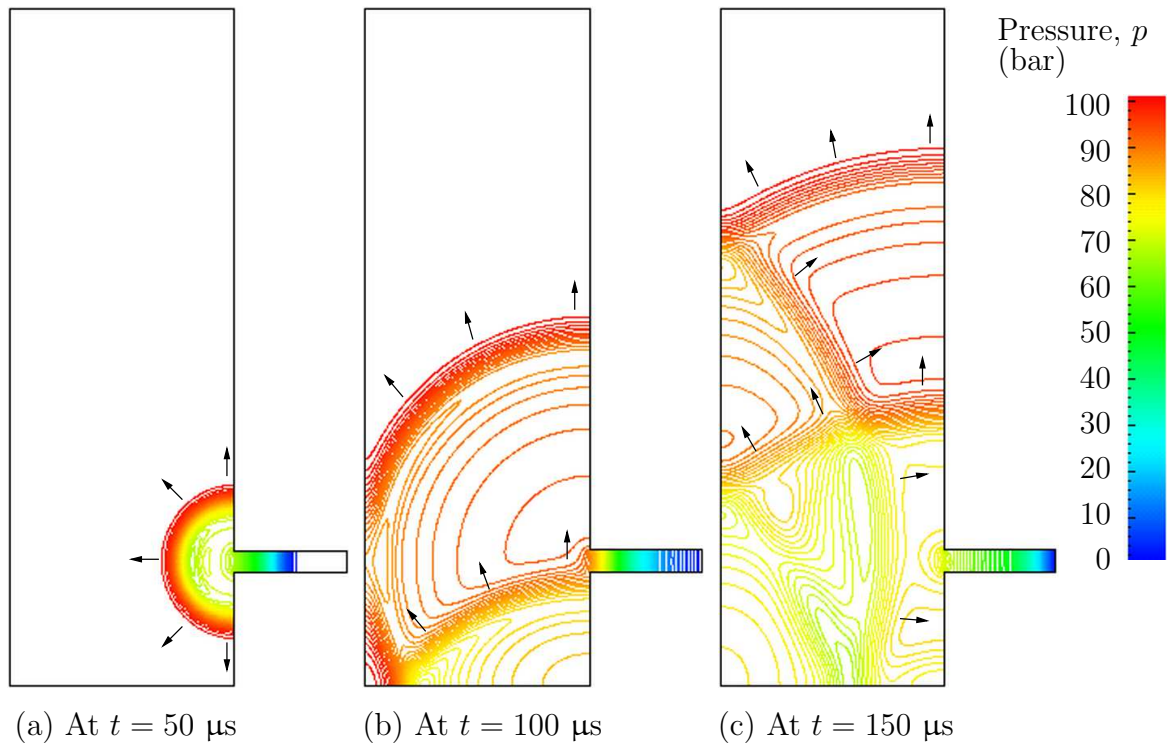


Figure 3.8: Propagation of pressure waves with refined mesh

edit the *blockMeshDict* and increase the number of cells by a factor of 4 in the  $x$  and  $y$  directions, *i.e.* block 0 becomes (120 80 1) from (30 20 1) and so on. Run **blockMesh** on this file. In addition, in order to maintain a Courant number below 1, the time step must be reduced accordingly to  $\Delta t = 10^{-7}$  s. The second simulation gives considerably better resolution of the pressure waves as shown in Figure 3.8.



# Chapter 4

## Multiphase flow

## 4.1 Breaking of a dam

Tutorial path:

- [\\$FOAM\\_TUTORIALS/multiphase/interFoam/laminar/damBreak/damBreak](#)

This example introduces the following OpenFOAM features for the first time:

- multiphase flow using VOF interface capturing method
- non-uniform initial conditions setup using `setFields` utility
- running case in parallel
- post-processing a case in parallel

### 4.1.1 Problem specification

In this tutorial we shall solve a problem of simplified dam break in 2 dimensions using the `interFoam`. The feature of the problem is a transient flow of two fluids separated by a sharp interface, or free surface. The two-phase algorithm in `interFoam` is based on the volume of fluid (VOF) method in which a specie transport equation is used to determine the relative volume fraction of the two phases, or phase fraction  $\alpha$ , in each computational cell. Physical properties are calculated as weighted averages based on this fraction. The nature of the VOF method means that an interface between the species is not explicitly computed, but rather emerges as a property of the phase fraction field. Since the phase fraction can have any value between 0 and 1, the interface is never sharply defined, but occupies a volume around the region where a sharp interface should exist.

The test setup consists of a column of water at rest located behind a membrane on the left side of a tank. At time  $t = 0$  s, the membrane is removed and the column of water collapses. During the collapse, the water impacts an obstacle at the bottom of the tank and creates a complicated flow structure, including several captured pockets of air. The geometry and the initial setup is shown in Figure 4.1.

### 4.1.2 Mesh generation

The user should go to the `damBreak` case in their `$FOAM.RUN/tutorials/multiphase/interFoam/laminar` directory. Generate the mesh running `blockMesh` as described previously. The `damBreak` mesh consist of 5 blocks; the `blockMeshDict` entries are given below.

```

17  scale    0.146;
18
19  vertices
20  (
21      (0 0 0)
22      (2 0 0)
23      (2.16438 0 0)
24      (4 0 0)
25      (0 0.32876 0)
26      (2 0.32876 0)
27      (2.16438 0.32876 0)
28      (4 0.32876 0)
29      (0 4 0)
30      (2 4 0)
31      (2.16438 4 0)
32      (4 4 0)
33      (0 0 0.1)
34      (2 0 0.1)
35      (2.16438 0 0.1)
36      (4 0 0.1)
37      (0 0.32876 0.1)
38      (2 0.32876 0.1)
39      (2.16438 0.32876 0.1)

```

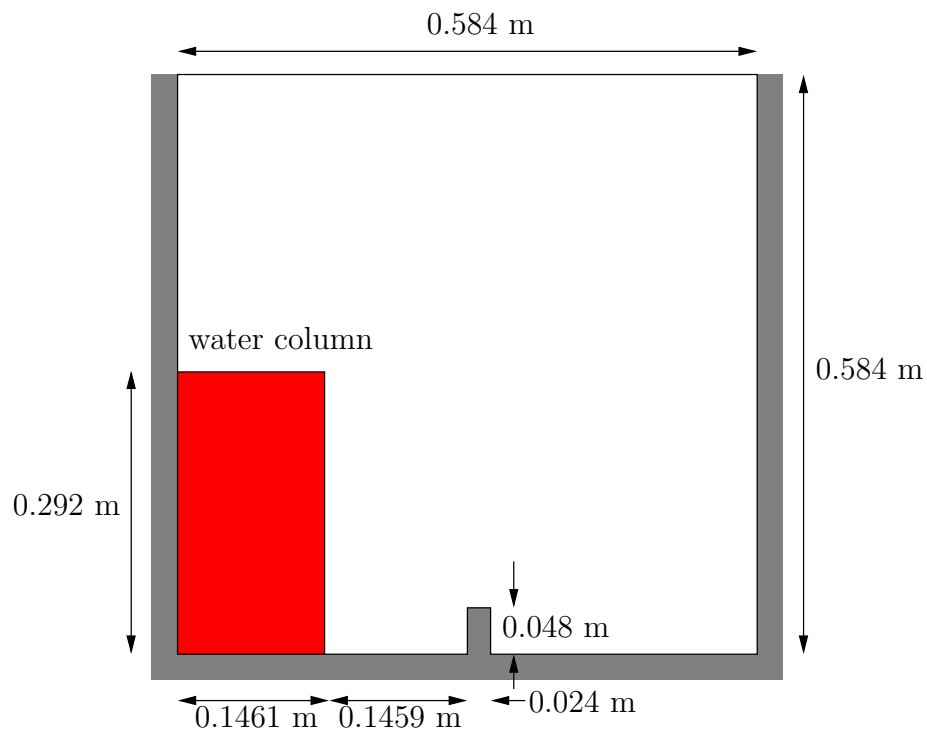


Figure 4.1: Geometry of the dam break.

```

40     (4 0.32876 0.1)
41     (0 4 0.1)
42     (2 4 0.1)
43     (2.16438 4 0.1)
44     (4 4 0.1)
45 );
46
47 blocks
48 (
49     hex (0 1 5 4 12 13 17 16) (23 8 1) simpleGrading (1 1 1)
50     hex (2 3 7 6 14 15 19 18) (19 8 1) simpleGrading (1 1 1)
51     hex (4 5 9 8 16 17 21 20) (23 42 1) simpleGrading (1 1 1)
52     hex (5 6 10 9 17 18 22 21) (4 42 1) simpleGrading (1 1 1)
53     hex (6 7 11 10 18 19 23 22) (19 42 1) simpleGrading (1 1 1)
54 );
55
56 edges
57 (
58 );
59
60 boundary
61 (
62     leftWall
63     {
64         type wall;
65         faces
66         (
67             (0 12 16 4)
68             (4 16 20 8)
69         );
70     }
71     rightWall
72     {
73         type wall;
74         faces
75         (
76             (7 19 15 3)
77             (11 23 19 7)
78         );
79     }
80     lowerWall
81     {
82         type wall;
83         faces
84         (
85             (0 1 13 12)
86             (1 5 17 13)
87             (5 6 18 17)

```

```

88             (2 14 18 6)
89             (2 3 15 14)
90         );
91     }
92     atmosphere
93     {
94         type patch;
95         faces
96         (
97             (8 20 21 9)
98             (9 21 22 10)
99             (10 22 23 11)
100        );
101    }
102 );
103
104 mergePatchPairs
105 (
106 );
107
108
109 // *****

```

### 4.1.3 Boundary conditions

The user can examine the boundary geometry generated by `blockMesh` by viewing the *boundary* file in the *constant/polyMesh* directory. The file contains a list of 5 boundary patches: `leftWall`, `rightWall`, `lowerWall`, `atmosphere` and `defaultFaces`. The user should notice the `type` of the patches. The `atmosphere` is a standard `patch`, *i.e.* has no special attributes, merely an entity on which boundary conditions can be specified. The `defaultFaces` patch is `empty` since the patch normal is in the direction we will not solve in this 2D case. The `leftWall`, `rightWall` and `lowerWall` patches are each a `wall`. Like the plain `patch`, the `wall` type contains no geometric or topological information about the mesh and only differs from the plain `patch` in that it identifies the patch as a wall, should an application need to know, *e.g.* to apply special wall surface modelling.

A good example is that the `interFoam` solver includes modelling of surface tension at the contact point between the interface and wall surface. The models are applied by specifying the `alphaContactAngle` boundary condition on the `alpha.water` ( $\alpha$ ) field. With it, the user must specify the following: a static contact angle, `theta0`  $\theta_0$ ; leading and trailing edge dynamic contact angles, `thetaA`  $\theta_A$  and `thetaR`  $\theta_R$  respectively; and a velocity scaling function for dynamic contact angle, `uTheta`.

In this tutorial we would like to ignore surface tension effects between the wall and interface. We can do this by setting the static contact angle,  $\theta_0 = 90^\circ$  and the velocity scaling function to 0. However, the simpler option which we shall choose here is to specify a `zeroGradient` type on `alpha.water`, rather than use the `alphaContactAngle` boundary condition.

The `top` boundary is free to the atmosphere so needs to permit both outflow and inflow according to the internal flow. We therefore use a combination of boundary conditions for pressure and velocity that does this while maintaining stability. They are:

- `totalPressure` which is a `fixedValue` condition calculated from specified total pressure `p0` and local velocity `U`;
- `pressureInletOutletVelocity`, which applies `zeroGradient` on all components, except where there is inflow, in which case a `fixedValue` condition is applied to the *tangential* component;
- `inletOutlet`, which is a `zeroGradient` condition when flow outwards, `fixedValue` when flow is inwards.

At all wall boundaries, the `fixedFluxPressure` boundary condition is applied to the pressure field, which calculates the normal gradient from the local density gradient.

The `defaultFaces` patch representing the front and back planes of the 2D problem, is, as usual, an `empty` type.

#### 4.1.4 Setting initial field

Unlike the previous cases, we shall now specify a non-uniform initial condition for the water phase fraction,  $\alpha$ , where

$$\alpha = \begin{cases} 1 & \text{for the liquid phase} \\ 0 & \text{for the gas phase} \end{cases} \quad (4.1)$$

This is achieved by running the `setFields` utility. It requires a `setFieldsDict` dictionary, located in the `system` directory, whose entries for this case are shown below.

```

17 defaultFieldValues
18 (
19     volScalarFieldValue alpha.water 0
20 );
21
22 regions
23 (
24     boxToCell
25     {
26         box (0 0 -1) (0.1461 0.292 1);
27         fieldValues
28         (
29             volScalarFieldValue alpha.water 1
30         );
31     }
32 );
33
34 // *****
35
```

The `defaultFieldValues` sets the default value of the fields, *i.e.* the value the field takes unless specified otherwise in the `regions` sub-dictionary. That sub-dictionary contains a list of subdictionaries containing `fieldValues` that override the defaults in a specified region. The region is expressed in terms of a `topoSetSource` that creates a set of points, cells or faces based on some topological constraint. Here, `boxToCell` creates a bounding box within a vector minimum and maximum to define the set of cells of the liquid region. The phase fraction  $\alpha$  is defined as 1 in this region.

The `setFields` utility reads fields from file and, after re-calculating those fields, will write them back to file. Because the files are then overridden, it is recommended that a backup is made before `setFields` is executed. In the `damBreak` tutorial, the `alpha.water` field is initially stored as a backup *only*, named `alpha.water.orig`. Before running `setFields`, the user first needs to copy `alpha.water.orig` to `alpha.water`, *e.g.* by typing:

```
cp 0/alpha.water.orig 0/alpha.water
```

The user should then execute `setFields` as any other utility is executed. Using `paraFoam`, check that the initial `alpha.water` field corresponds to the desired distribution as in Figure 4.2.

#### 4.1.5 Fluid properties

Let us examine the `transportProperties` file in the `constant` directory. Its dictionary contains the material properties for each fluid, separated into two subdictionaries `phase1`

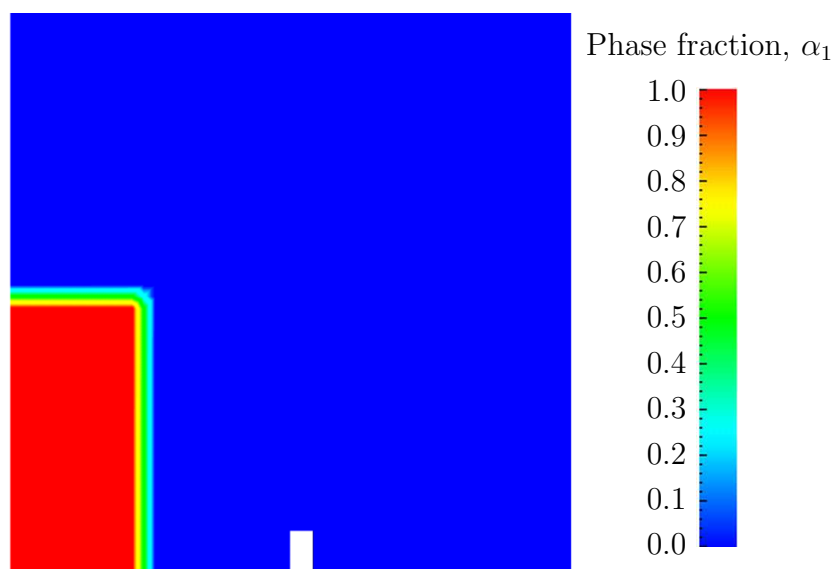


Figure 4.2: Initial conditions for phase fraction `alpha.water`.

and *phase2*. The transport model for each phase is selected by the `transportModel` keyword. The user should select `Newtonian` in which case the kinematic viscosity is single valued and specified under the keyword `nu`. The viscosity parameters for the other models, *e.g.* `CrossPowerLaw`, are specified within subdictionaries with the generic name `<model>Coeffs`, *i.e.* `CrossPowerLawCoeffs` in this example. The density is specified under the keyword `rho`.

The surface tension between the two phases is specified under the keyword `sigma`. The values used in this tutorial are listed in Table 4.1.

phase1 properties			
Kinematic viscosity	$\text{m}^2 \text{s}^{-1}$	<code>nu</code>	$1.0 \times 10^{-6}$
Density	$\text{kg m}^{-3}$	<code>rho</code>	$1.0 \times 10^3$
phase2 properties			
Kinematic viscosity	$\text{m}^2 \text{s}^{-1}$	<code>nu</code>	$1.48 \times 10^{-5}$
Density	$\text{kg m}^{-3}$	<code>rho</code>	1.0
Properties of both phases			
Surface tension	$\text{N m}^{-1}$	<code>sigma</code>	0.07

Table 4.1: Fluid properties for the `damBreak` tutorial

Gravitational acceleration is uniform across the domain and is specified in a file named *g* in the *constant* directory. Unlike a normal field file, *e.g.* *U* and *p*, *g* is a `uniformDimensionedVectorField` and so simply contains a set of `dimensions` and a `value` that represents  $(0, 9.81, 0) \text{ m s}^{-2}$  for this tutorial:

```

17 dimensions      [0 1 -2 0 0 0 0];
18 value           (0 -9.81 0);
19
20
21 // ***** //
```



### 4.1.6 Turbulence modelling

As in the cavity example, the choice of turbulence modelling method is selectable at run-time through the `simulationType` keyword in *turbulenceProperties* dictionary. In this example, we wish to run without turbulence modelling so we set `laminar`:

```

17  simulationType  laminar;
18
19
20  // ***** //
```

### 4.1.7 Time step control

Time step control is an important issue in free surface tracking since the surface-tracking algorithm is considerably more sensitive to the Courant number  $Co$  than in standard fluid flow calculations. Ideally, we should not exceed an upper limit  $Co \approx 0.5$  in the region of the interface. In some cases, where the propagation velocity is easy to predict, the user should specify a fixed time-step to satisfy the  $Co$  criterion. For more complex cases, this is considerably more difficult. *interFoam* therefore offers automatic adjustment of the time step as standard in the *controlDict*. The user should specify `adjustTimeStep` to be `yes` and the maximum  $Co$  for the phase fields, `maxAlphaCo`, and other fields, `maxCo`, to be 0.5. The upper limit on time step `maxDeltaT` can be set to a value that will not be exceeded in this simulation, *e.g.* 1.0.

By using automatic time step control, the steps themselves are never rounded to a convenient value. Consequently if we request that OpenFOAM saves results at a fixed number of time step intervals, the times at which results are saved are somewhat arbitrary. However even with automatic time step adjustment, OpenFOAM allows the user to specify that results are written at fixed times; in this case OpenFOAM forces the automatic time stepping procedure to adjust time steps so that it ‘hits’ on the exact times specified for write output. The user selects this with the `adjustableRunTime` option for `writeControl` in the *controlDict* dictionary. The *controlDict* dictionary entries should be:

```

17  application      interFoam;
18
19  startFrom         startTime;
20
21  startTime         0;
22
23  stopAt            endTime;
24
25  endTime           1;
26
27  deltaT            0.001;
28
29  writeControl       adjustable;
30
31  writeInterval      0.05;
32
33  purgeWrite         0;
34
35  writeFormat        ascii;
36
37  writePrecision     6;
38
39  writeCompression  off;
40
41  timeFormat         general;
42
43  timePrecision      6;
44
45  runtimeModifiable yes;
46
47  adjustTimeStep     yes;
48
49  maxCo              1;
50
51  maxAlphaCo         1;
52
53  maxDeltaT          1;
```

```

54
55 functions
56 {
57     #include    "sampling"
58 }
59
60
61 // ***** //

```

### 4.1.8 Discretisation schemes

The `interFoam` solver uses the multidimensional universal limiter for explicit solution (MULES) method, created by OpenCFD, to maintain boundedness of the phase fraction independent of underlying numerical scheme, mesh structure, *etc.* The choice of schemes for convection are therefore not restricted to those that are strongly stable or bounded, *e.g.* upwind differencing.

The convection schemes settings are made in the `divSchemes` sub-dictionary of the `fvSchemes` dictionary. In this example, the convection term in the momentum equation ( $\nabla \cdot (\rho \mathbf{U} \mathbf{U})$ ), denoted by the `div(rhoPhi,U)` keyword, uses `Gauss linearUpwind grad(U)` to produce good accuracy. The  $\nabla \cdot (\mathbf{U} \alpha)$  term, represented by the `div(phi,alpha)` keyword uses the `vanLeer` scheme. The  $\nabla \cdot (\mathbf{U}_{rb} \alpha)$  term, represented by the `div(phirb,alpha)` keyword, can similarly use the `vanLeer` scheme, but generally produces smoother interfaces using the `linear` scheme.

The other discretised terms use commonly employed schemes so that the `fvSchemes` dictionary entries should therefore be:

```

17 ddtSchemes
18 {
19     default      Euler;
20 }
21
22 gradSchemes
23 {
24     default      Gauss linear;
25 }
26
27 divSchemes
28 {
29     div(rhoPhi,U)  Gauss linearUpwind grad(U);
30     div(phi,alpha) Gauss vanLeer;
31     div(phirb,alpha) Gauss linear;
32     div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
33 }
34
35 laplacianSchemes
36 {
37     default      Gauss linear corrected;
38 }
39
40 interpolationSchemes
41 {
42     default      linear;
43 }
44
45 snGradSchemes
46 {
47     default      corrected;
48 }
49
50
51 // ***** //

```

### 4.1.9 Linear-solver control

In the `fvSolution`, the `PISO` sub-dictionary contains elements that are specific to `interFoam`. There are the usual correctors to the momentum equation but also correctors to a PISO loop around the  $\alpha$  phase equation. Of particular interest are the `nAlphaSubCycles` and `cAlpha` keywords. `nAlphaSubCycles` represents the number of sub-cycles within the  $\alpha$

equation; sub-cycles are additional solutions to an equation within a given time step. It is used to enable the solution to be stable without reducing the time step and vastly increasing the solution time. Here we specify 2 sub-cycles, which means that the  $\alpha$  equation is solved in  $2\times$  half length time steps within each actual time step.

The `cAlpha` keyword is a factor that controls the compression of the interface where: 0 corresponds to no compression; 1 corresponds to conservative compression; and, anything larger than 1, relates to enhanced compression of the interface. We generally recommend a value of 1.0 which is employed in this example.

#### 4.1.10 Running the code

Running of the code has been described in detail in previous tutorials. Try the following, that uses `tee`, a command that enables output to be written to both standard output and files:

```
cd $FOAM_RUN/tutorials/multiphase/interFoam/laminar/damBreak/damBreak
interFoam | tee log
```

The code will now be run interactively, with a copy of output stored in the `log` file.

#### 4.1.11 Post-processing

Post-processing of the results can now be done in the usual way. The user can monitor the development of the phase fraction `alpha.water` in time, *e.g.* see Figure 4.3.

#### 4.1.12 Running in parallel

The results from the previous example are generated using a fairly coarse mesh. We now wish to increase the mesh resolution and re-run the case. The new case will typically take a few hours to run with a single processor so, should the user have access to multiple processors, we can demonstrate the parallel processing capability of OpenFOAM.

The user should first make a copy of the `damBreak` case, *e.g.* by

```
cd $FOAM_RUN/tutorials/multiphase/interFoam/laminar/damBreak
mkdir damBreakFine
cp -r damBreak/0 damBreakFine
cp -r damBreak/system damBreakFine
cp -r damBreak/constant damBreakFine
```

Enter the new case directory and change the `blocks` description in the `blockMeshDict` dictionary to

```
blocks
(
    hex (0 1 5 4 12 13 17 16) (46 10 1) simpleGrading (1 1 1)
    hex (2 3 7 6 14 15 19 18) (40 10 1) simpleGrading (1 1 1)
    hex (4 5 9 8 16 17 21 20) (46 76 1) simpleGrading (1 2 1)
    hex (5 6 10 9 17 18 22 21) (4 76 1) simpleGrading (1 2 1)
    hex (6 7 11 10 18 19 23 22) (40 76 1) simpleGrading (1 2 1)
);
```

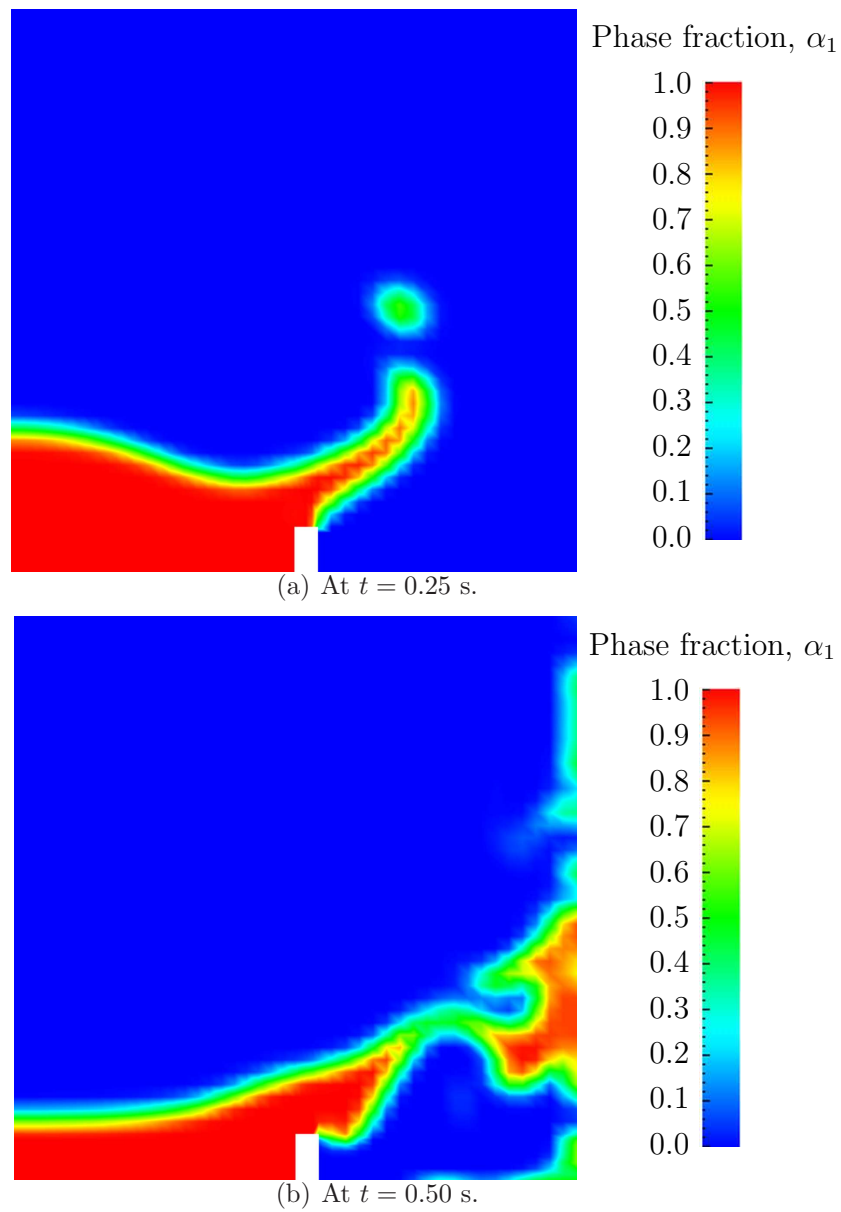


Figure 4.3: Snapshots of liquid phase  $\alpha$ .

Here, the entry is presented as printed from the *blockMeshDict* file; in short the user must change the mesh densities, *e.g.* the 46 10 1 entry, and some of the mesh grading entries to 1 2 1. Once the dictionary is correct, generate the mesh.

As the mesh has now changed from the **damBreak** example, the user must re-initialise the phase field **alpha.water** in the 0 time directory since it contains a number of elements that is inconsistent with the new mesh. Note that there is no need to change the **U** and **p\_rgh** fields since they are specified as **uniform** which is independent of the number of elements in the field. We wish to initialise the field with a sharp interface, *i.e.* it elements would have  $\alpha = 1$  or  $\alpha = 0$ . Updating the field with **mapFields** may produce interpolated values  $0 < \alpha < 1$  at the interface, so it is better to rerun the **setFields** utility. There is a backup copy of the initial uniform  $\alpha$  field named *0/alpha.water.org* that the user should copy to *0/alpha.water* before running **setFields**:

```
cd $FOAM_RUN/tutorials/multiphase/interFoam/laminar/damBreak/damBreakFine
cp -r 0/alpha.water.org 0/alpha.water
setFields
```

The method of parallel computing used by OpenFOAM is known as domain decomposition, in which the geometry and associated fields are broken into pieces and allocated to separate processors for solution. The first step required to run a parallel case is therefore to decompose the domain using the **decomposePar** utility. There is a dictionary associated with **decomposePar** named *decomposeParDict* which is located in the **system** directory of the tutorial case; also, like with many utilities, a default dictionary can be found in the directory of the source code of the specific utility, *i.e.* in *\$FOAM\_UTILITIES/parallelProcessing/decomposePar* for this case.

The first entry is **numberOfSubdomains** which specifies the number of subdomains into which the case will be decomposed, usually corresponding to the number of processors available for the case.

In this tutorial, the **method** of decomposition should be **simple** and the corresponding **simpleCoeffs** should be edited according to the following criteria. The domain is split into pieces, or subdomains, in the  $x$ ,  $y$  and  $z$  directions, the number of subdomains in each direction being given by the vector **n**. As this geometry is 2 dimensional, the 3rd direction,  $z$ , cannot be split, hence  $n_z$  must equal 1. The  $n_x$  and  $n_y$  components of **n** split the domain in the  $x$  and  $y$  directions and must be specified so that the number of subdomains specified by  $n_x$  and  $n_y$  equals the specified **numberOfSubdomains**, *i.e.*  $n_x n_y = \text{numberOfSubdomains}$ . It is beneficial to keep the number of cell faces adjoining the subdomains to a minimum so, for a square geometry, it is best to keep the split between the  $x$  and  $y$  directions should be fairly even. The **delta** keyword should be set to 0.001.

For example, let us assume we wish to run on 4 processors. We would set **numberOfSubdomains** to 4 and **n** = (2, 2, 1). When running **decomposePar**, we can see from the screen messages that the decomposition is distributed fairly even between the processors.

The user should consult User Guide section 3.2 for details of how to run a case in parallel; in this tutorial we merely present an example of running in parallel. We use the **openMPI** implementation of the standard message-passing interface (MPI). As a test here, the user can run in parallel on a single node, the local host only, by typing:

```
mpirun -np 4 interFoam -parallel > log &
```

The user may run on more nodes over a network by creating a file that lists the host names of the machines on which the case is to be run as described in User Guide

section 3.2.2. The case should run in the background and the user can follow its progress by monitoring the *log* file as usual.

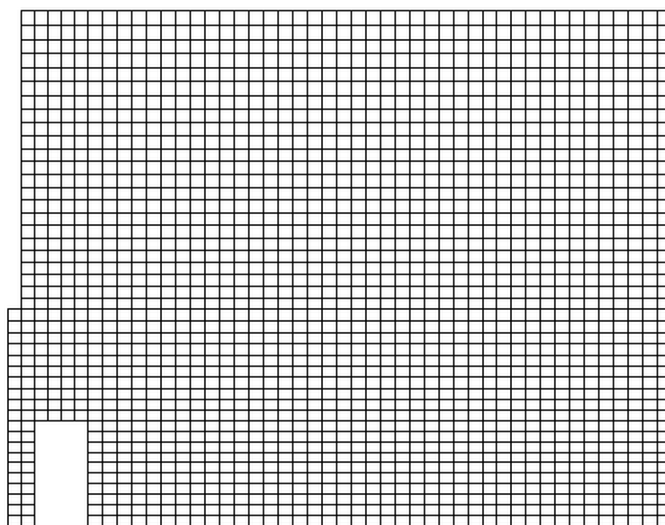


Figure 4.4: Mesh of processor 2 in parallel processed case.

#### 4.1.13 Post-processing a case run in parallel

Once the case has completed running, the decomposed fields and mesh must be reassembled for post-processing using the `reconstructPar` utility. Simply execute it from the command line. The results from the fine mesh are shown in Figure 4.5. The user can see that the resolution of interface has improved significantly compared to the coarse mesh.

The user may also post-process a segment of the decomposed domain individually by simply treating the individual processor directory as a case in its own right. For example if the user starts `paraFoam` by

```
paraFoam -case processor1
```

then `processor1` will appear as a case module in `ParaView`. Figure 4.4 shows the mesh from processor 1 following the decomposition of the domain using the `simple` method.

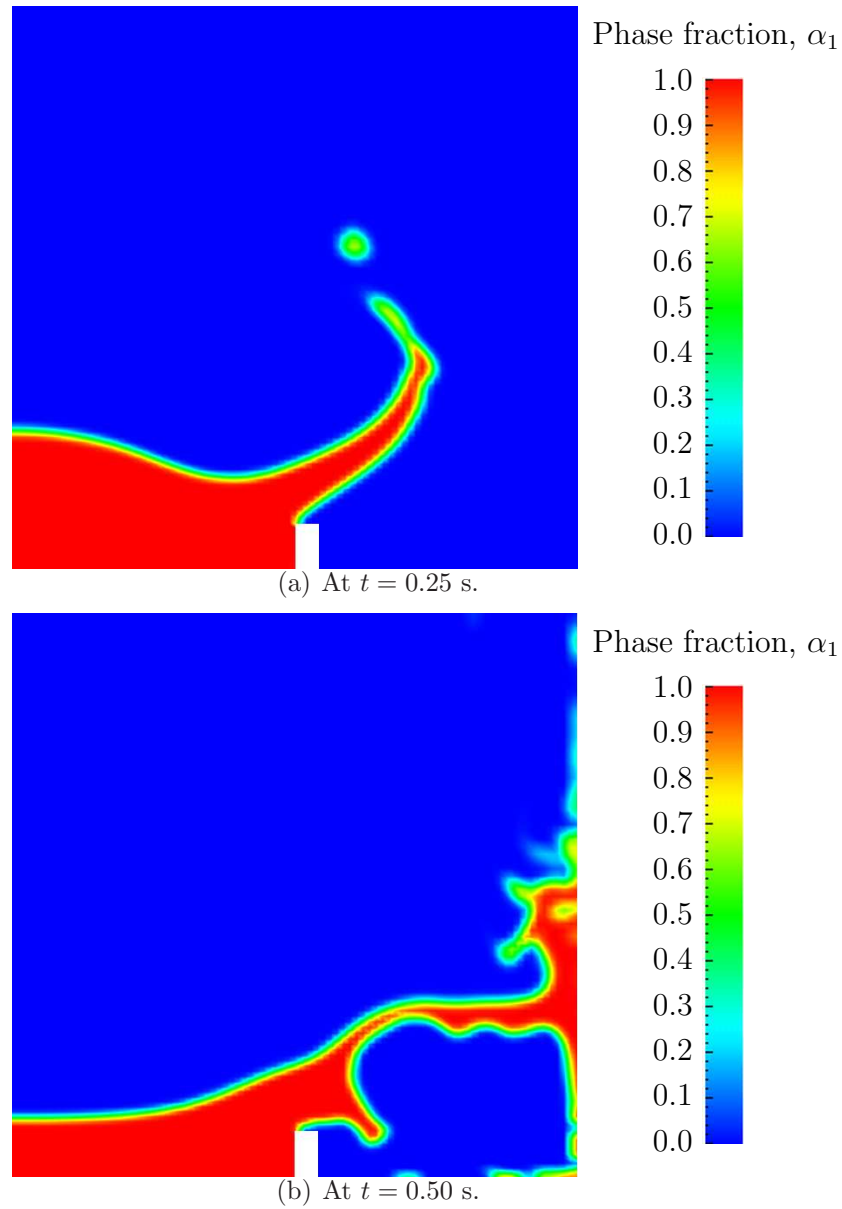


Figure 4.5: Snapshots of liquid phase  $\alpha$  with refined mesh.





# Chapter 5

## Stress analysis

## 5.1 Stress analysis of a plate with a hole

Tutorial path:

- [\\$FOAM\\_TUTORIALS/stressAnalysis/solidDisplacementFoam/plateHole](#)

This tutorial describes how to pre-process, run and post-process a case involving linear-elastic, steady-state stress analysis on a square plate with a circular hole at its centre.

- thermal stress analysis using `solidDisplacementFoam` solver
- validation against the analytical solution

The plate dimensions are: side length 4 m and radius  $R = 0.5$  m. It is loaded with a uniform traction of  $\sigma = 10$  kPa over its left and right faces as shown in Figure 5.1. Two symmetry planes can be identified for this geometry and therefore the solution domain need only cover a quarter of the geometry, shown by the shaded area in Figure 5.1.

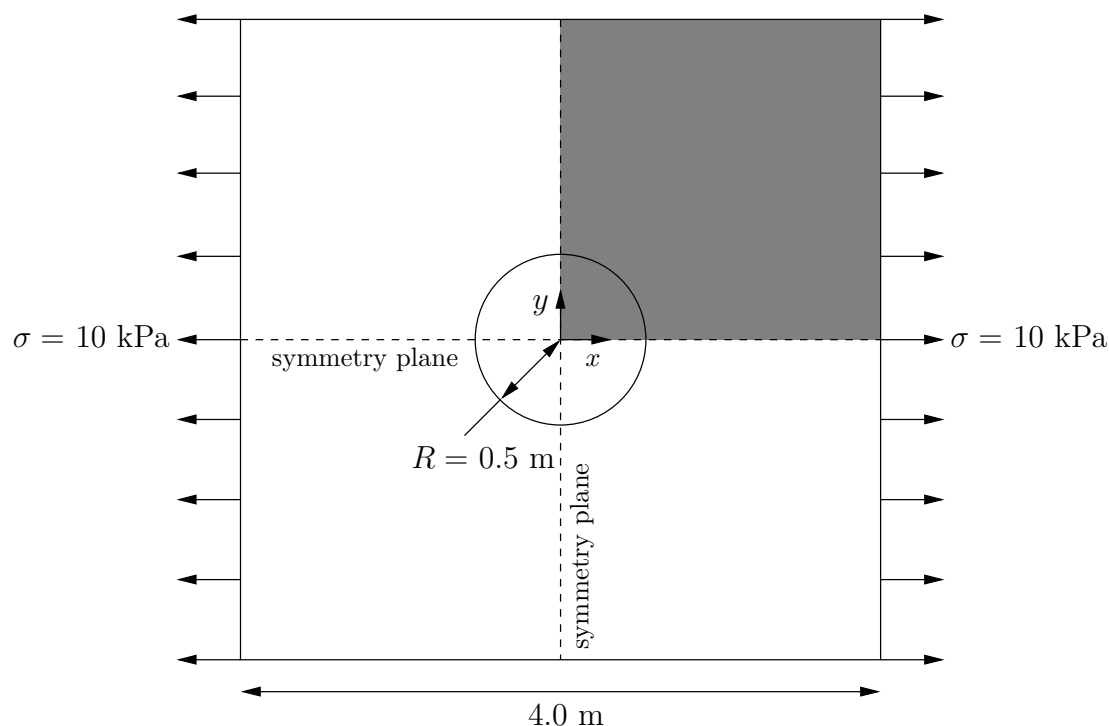


Figure 5.1: Geometry of the plate with a hole.

### 5.1.1 Problem specification

The problem can be approximated as 2-dimensional since the load is applied in the plane of the plate. In a Cartesian coordinate system there are two possible assumptions to take in regard to the behaviour of the structure in the third dimension: (1) the plane stress condition, in which the stress components acting out of the 2D plane are assumed to be negligible; (2) the plane strain condition, in which the strain components out of the 2D plane are assumed negligible. The plane stress condition is appropriate for solids whose third dimension is thin as in this case; the plane strain condition is applicable for solids where the third dimension is thick.

An analytical solution exists for loading of an infinitely large, thin plate with a circular hole. The solution for the stress normal to the vertical plane of symmetry is

$$(\sigma_{xx})_{x=0} = \begin{cases} \sigma \left( 1 + \frac{R^2}{2y^2} + \frac{3R^4}{2y^4} \right) & \text{for } |y| \geq R \\ 0 & \text{for } |y| < R \end{cases} \quad (5.1)$$

Results from the simulation will be compared with this solution. At the end of the tutorial, the user can: investigate the sensitivity of the solution to mesh resolution and mesh grading; and, increase the size of the plate in comparison to the hole to try to estimate the error in comparing the analytical solution for an infinite plate to the solution of this problem of a finite plate.

### 5.1.2 Mesh generation

The domain consists of four blocks, some of which have arc-shaped edges. The block structure for the part of the mesh in the  $x - y$  plane is shown in Figure 5.2. As already mentioned in section 2.1.1.1, all geometries are generated in 3 dimensions in OpenFOAM even if the case is to be as a 2 dimensional problem. Therefore a dimension of the block in the  $z$  direction has to be chosen; here, 0.5 m is selected. It does not affect the solution since the traction boundary condition is specified as a stress rather than a force, thereby making the solution independent of the cross-sectional area.

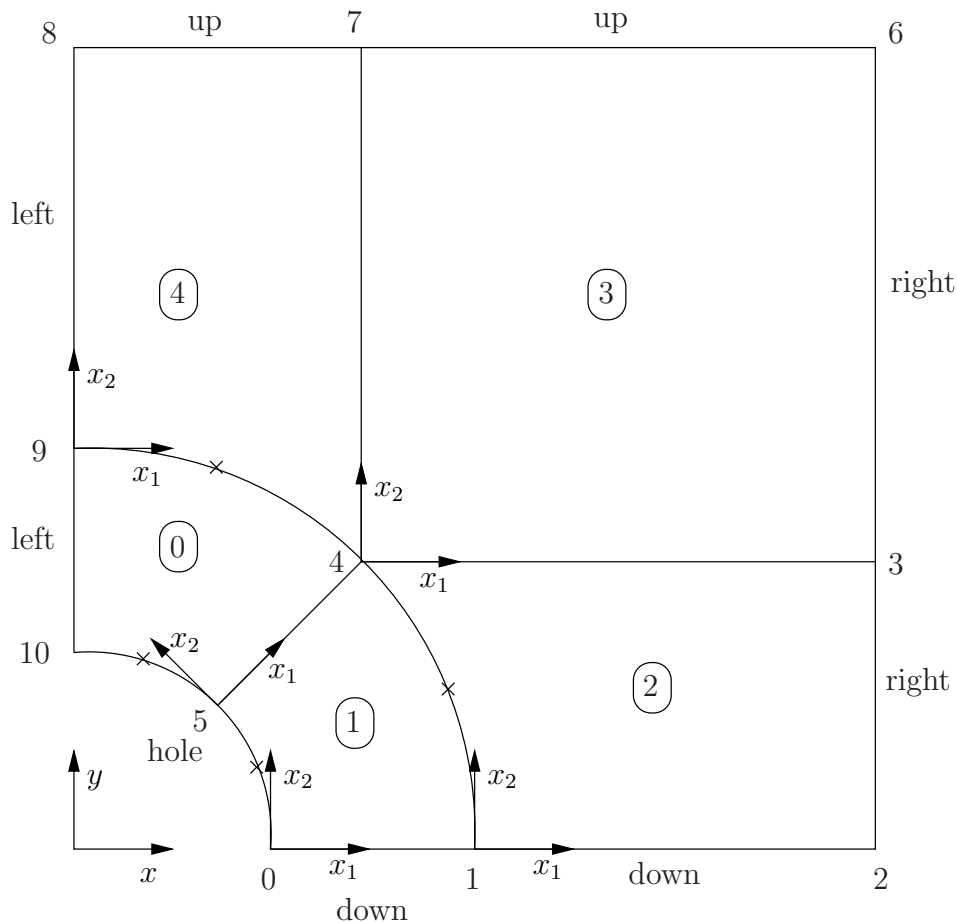


Figure 5.2: Block structure of the mesh for the plate with a hole.

The user should change into the `plateHole` case in the `$FOAM_RUN/tutorials/stress-Analysis/solidDisplacementFoam` directory and open the `system/blockMeshDict` file in an editor, as listed below

```

17  scale 1;
18
19  vertices
20  (
21      (0.5 0 0)
22      (1 0 0)
23      (2 0 0)
24      (2 0.707107 0)
25      (0.707107 0.707107 0)
26      (0.353553 0.353553 0)
27      (2 2 0)
28      (0.707107 2 0)
29      (0 2 0)
30      (0 1 0)
31      (0 0.5 0)
32      (0.5 0 0.5)
33      (1 0 0.5)
34      (2 0 0.5)
35      (2 0.707107 0.5)
36      (0.707107 0.707107 0.5)
37      (0.353553 0.353553 0.5)
38      (2 2 0.5)
39      (0.707107 2 0.5)
40      (0 2 0.5)
41      (0 1 0.5)
42      (0 0.5 0.5)
43  );
44
45  blocks
46  (
47      hex (5 4 9 10 16 15 20 21) (10 10 1) simpleGrading (1 1 1)
48      hex (0 1 4 5 11 12 15 16) (10 10 1) simpleGrading (1 1 1)
49      hex (1 2 3 4 12 13 14 15) (20 10 1) simpleGrading (1 1 1)
50      hex (4 3 6 7 15 14 17 18) (20 20 1) simpleGrading (1 1 1)
51      hex (9 4 7 8 20 15 18 19) (10 20 1) simpleGrading (1 1 1)
52  );
53
54  edges
55  (
56      arc 0 5 origin (0 0 0)
57      arc 5 10 origin (0 0 0)
58      arc 1 4 origin (0 0 0)
59      arc 4 9 origin (0 0 0)
60      arc 11 16 origin (0 0 0.5)
61      arc 16 21 origin (0 0 0.5)
62      arc 12 15 origin (0 0 0.5)
63      arc 15 20 origin (0 0 0.5)
64  );
65
66  boundary
67  (
68      left
69      {
70          type symmetryPlane;
71          faces
72          (
73              (8 9 20 19)
74              (9 10 21 20)
75          );
76      }
77      right
78      {
79          type patch;
80          faces
81          (
82              (2 3 14 13)
83              (3 6 17 14)
84          );
85      }
86      down
87      {
88          type symmetryPlane;
89          faces
90          (
91              (0 1 12 11)
92              (1 2 13 12)
93          );
94      }
95      up
96      {
97          type patch;
98          faces
99          (

```

```

100             (7 8 19 18)
101             (6 7 18 17)
102         );
103     }
104     hole
105     {
106         type patch;
107         faces
108         (
109             (10 5 16 21)
110             (5 0 11 16)
111         );
112     }
113     frontAndBack
114     {
115         type empty;
116         faces
117         (
118             (10 9 4 5)
119             (5 4 1 0)
120             (1 4 3 2)
121             (4 7 6 3)
122             (4 9 8 7)
123             (21 16 15 20)
124             (16 11 12 15)
125             (12 13 14 15)
126             (15 14 17 18)
127             (15 18 19 20)
128         );
129     }
130 );
131
132
133 // *****

```

Until now, we have only specified straight edges in the geometries of previous tutorials but here we need to specify curved edges. These are specified under the `edges` keyword entry which is a list of non-straight edges. The syntax of each list entry begins with the type of curve, including `arc`, `simpleSpline`, `polyLine` *etc.*, described further in User Guide section 4.3.1. In this example, all the edges are circular and so can be specified by the `arc` keyword entry. The following entries are the labels of the start and end vertices of the arc and a point vector through which the circular arc passes.

The blocks in this *blockMeshDict* do not all have the same orientation. As can be seen in Figure 5.2 the  $x_2$  direction of block 0 is equivalent to the  $-x_1$  direction for block 4. This means care must be taken when defining the number and distribution of cells in each block so that the cells match up at the block faces.

6 patches are defined: one for each side of the plate, one for the hole and one for the front and back planes. The `left` and `down` patches are both a symmetry plane. Since this is a *geometric* constraint, it is included in the definition of the *mesh*, rather than being purely a specification on the boundary condition of the fields. Therefore they are defined as such using a special `symmetryPlane` type as shown in the *blockMeshDict*.

The `frontAndBack` patch represents the plane which is ignored in a 2D case. Again this is a geometric constraint so is defined within the mesh, using the `empty` type as shown in the *blockMeshDict*. For further details of boundary types and geometric constraints, the user should refer to User Guide section 4.2.1.

The remaining patches are of the regular `patch` type. The mesh should be generated using `blockMesh` and can be viewed in `paraFoam` as described in section 2.1.2. It should appear as in Figure 5.3.

### 5.1.2.1 Boundary and initial conditions

Once the mesh generation is complete, the initial field with boundary conditions must be set. For a stress analysis case without thermal stresses, only displacement `D` needs to be set. The *0.orig/D* is as follows:

```

17     dimensions      [0 1 0 0 0 0 0];
18

```

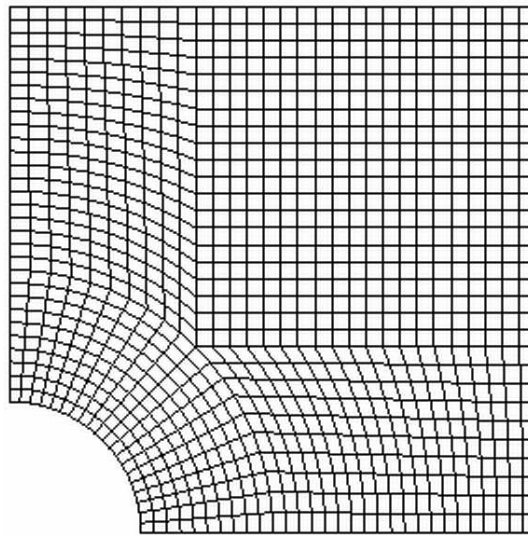


Figure 5.3: Mesh of the hole in a plate problem.

```

19  internalField    uniform (0 0 0);
20
21  boundaryField
22  {
23      left
24      {
25          type      symmetryPlane;
26      }
27
28      right
29      {
30          type      tractionDisplacement;
31          traction   uniform (10000 0 0);
32          pressure   uniform 0;
33          value      uniform (0 0 0);
34      }
35
36      down
37      {
38          type      symmetryPlane;
39      }
40
41      up
42      {
43          type      tractionDisplacement;
44          traction   uniform (0 0 0);
45          pressure   uniform 0;
46          value      uniform (0 0 0);
47      }
48
49      hole
50      {
51          type      tractionDisplacement;
52          traction   uniform (0 0 0);
53          pressure   uniform 0;
54          value      uniform (0 0 0);
55      }
56
57      frontAndBack
58      {
59          type      empty;
60      }
61  }
62
63
64  // *****

```

Firstly, it can be seen that the displacement initial conditions are set to  $(0, 0, 0)$  m. The **left** and **down** patches **must** be both of `symmetryPlane` type since they are specified as such in the mesh description in the `constant/polyMesh/boundary` file. Similarly the **frontAndBack** patch is declared `empty`.

The other patches are traction boundary conditions, set by a specialist `traction` bound-

ary type. The traction boundary conditions are specified by a linear combination of: (1) a boundary traction vector under keyword **traction**; (2) a pressure that produces a traction normal to the boundary surface that is defined as negative when pointing out of the surface, under keyword **pressure**. The **up** and **hole** patches are zero traction so the boundary traction and pressure are set to zero. For the **right** patch the traction should be  $(1e4, 0, 0)$  Pa and the pressure should be 0 Pa.

### 5.1.2.2 Mechanical properties

The physical properties for the case are set in the *mechanicalProperties* dictionary in the *constant* directory. For this problem, we need to specify the mechanical properties of steel given in Table 5.1. In the mechanical properties dictionary, the user must also set **planeStress** to **yes**.

Property	Units	Keyword	Value
Density	$\text{kg m}^{-3}$	<b>rho</b>	7854
Young's modulus	Pa	<b>E</b>	$2 \times 10^{11}$
Poisson's ratio	—	<b>nu</b>	0.3

Table 5.1: Mechanical properties for steel

### 5.1.2.3 Thermal properties

The temperature field variable **T** is present in the **solidDisplacementFoam** solver since the user may opt to solve a thermal equation that is coupled with the momentum equation through the thermal stresses that are generated. The user specifies at run time whether OpenFOAM should solve the thermal equation by the **thermalStress** switch in the *thermalProperties* dictionary. This dictionary also sets the thermal properties for the case, *e.g.* for steel as listed in Table 5.2.

Property	Units	Keyword	Value
Specific heat capacity	$\text{J kg}^{-1} \text{K}^{-1}$	<b>C</b>	434
Thermal conductivity	$\text{W m}^{-1} \text{K}^{-1}$	<b>k</b>	60.5
Thermal expansion coeff.	$\text{K}^{-1}$	<b>alpha</b>	$1.1 \times 10^{-5}$

Table 5.2: Thermal properties for steel

In this case we do not want to solve for the thermal equation. Therefore we must set the **thermalStress** keyword entry to **no** in the *thermalProperties* dictionary.

### 5.1.2.4 Control

As before, the information relating to the control of the solution procedure are read in from the *controlDict* dictionary. For this case, the **startTime** is 0 s. The time step is not important since this is a steady state case; in this situation it is best to set the time step **deltaT** to 1 so it simply acts as an iteration counter for the steady-state case. The **endTime**, set to 100, then acts as a limit on the number of iterations. The **writeInterval** can be set to 20.

The *controlDict* entries are as follows:

```

17 application      solidDisplacementFoam;
18
19 startFrom        startTime;
20
21 startTime        0;
22
23 stopAt           endTime;
24
25 endTime          100;
26
27 deltaT           1;
28
29 writeControl      timeStep;
30
31 writeInterval     20;
32
33 purgeWrite        0;
34
35 writeFormat       ascii;
36
37 writePrecision    6;
38
39 writeCompression  off;
40
41 timeFormat        general;
42
43 timePrecision     6;
44
45 graphFormat       raw;
46
47 runTimeModifiable true;
48
49
50 // ***** //

```

### 5.1.2.5 Discretisation schemes and linear-solver control

Let us turn our attention to the *fvSchemes* dictionary. Firstly, the problem we are analysing is steady-state so the user should select **SteadyState** for the time derivatives in **timeScheme**. This essentially switches off the time derivative terms. Not all solvers, especially in fluid dynamics, work for both steady-state and transient problems but **solidDisplacementFoam** does work, since the base algorithm is the same for both types of simulation.

The momentum equation in linear-elastic stress analysis includes several explicit terms containing the gradient of displacement. The calculations benefit from accurate and smooth evaluation of the gradient. Normally, in the finite volume method the discretisation is based on Gauss's theorem. The Gauss method is sufficiently accurate for most purposes but, in this case, the least squares method will be used. The user should therefore open the *fvSchemes* dictionary in the *system* directory and ensure the **leastSquares** method is selected for the **grad(U)** gradient discretisation scheme in the **gradSchemes** sub-dictionary:

```

17 d2dt2Schemes
18 {
19     default      steadyState;
20 }
21
22 ddtSchemes
23 {
24     default      Euler;
25 }
26
27 gradSchemes
28 {
29     default      leastSquares;
30     grad(D)       leastSquares;
31     grad(T)       leastSquares;
32 }
33
34 divSchemes
35 {
36     default      none;
37     div(sigmaD)  Gauss linear;
38 }

```



```

39
40  laplacianSchemes
41  {
42      default          none;
43      laplacian(DD,D) Gauss linear corrected;
44      laplacian(DT,T) Gauss linear corrected;
45  }
46
47  interpolationSchemes
48  {
49      default          linear;
50  }
51
52  snGradSchemes
53  {
54      default          none;
55  }
56
57
58  // *****

```

The *fvSolution* dictionary in the *system* directory controls the linear equation solvers and algorithms used in the solution. The user should first look at the *solvers* sub-dictionary and notice that the choice of *solver* for D is GAMG. The solver *tolerance* should be set to  $10^{-6}$  for this problem. The solver relative tolerance, denoted by *relTol*, sets the required reduction in the residuals within each iteration. It is uneconomical to set a tight (low) relative tolerance within each iteration since a lot of terms in each equation are explicit and are updated as part of the segregated iterative procedure. Therefore a reasonable value for the relative tolerance is 0.01, or possibly even higher, say 0.1, or in some cases even 0.9 (as in this case).

```

17  solvers
18  {
19      "(D|T)"
20      {
21          solver          GAMG;
22          tolerance       1e-06;
23          relTol          0.9;
24          smoother        GaussSeidel;
25          nCellsInCoarsestLevel 20;
26      }
27  }
28
29  stressAnalysis
30  {
31      compactNormalStress yes;
32      nCorrectors         1;
33      D                   1e-06;
34  }
35
36
37  // *****

```

The *fvSolution* dictionary contains a sub-dictionary, *stressAnalysis* that contains some control parameters specific to the application solver. Firstly there is *nCorrectors* which specifies the number of outer loops around the complete system of equations, including traction boundary conditions *within each time step*. Since this problem is steady-state, we are performing a set of iterations towards a converged solution with the 'time step' acting as an iteration counter. We can therefore set *nCorrectors* to 1.

The D keyword specifies a convergence tolerance for the outer iteration loop, *i.e.* sets a level of initial residual below which solving will cease. It should be set to the desired solver tolerance specified earlier,  $10^{-6}$  for this problem.

### 5.1.3 Running the code

The user should run the code here in the background from the command line as specified below, so he/she can look at convergence information in the log file afterwards.

```
cd $FOAM_RUN/tutorials/stressAnalysis/solidDisplacementFoam/plateHole
solidDisplacementFoam > log &
```

The user should check the convergence information by viewing the generated *log* file which shows the number of iterations and the initial and final residuals of the displacement in each direction being solved. The final residual should always be less than 0.9 times the initial residual as this iteration tolerance set. Once both initial residuals have dropped below the convergence tolerance of  $10^{-6}$  the run has converged and can be stopped by killing the batch job.

### 5.1.4 Post-processing

Post processing can be performed as in section 2.1.4. The `solidDisplacementFoam` solver outputs the stress field  $\sigma$  as a symmetric tensor field `sigma`. This is consistent with the way variables are usually represented in OpenFOAM solvers by the mathematical symbol by which they are represented; in the case of Greek symbols, the variable is named phonetically.

For post-processing individual scalar field components,  $\sigma_{xx}$ ,  $\sigma_{xy}$  etc., can be generated by running the `postProcess` utility as before in section 2.1.5.7, this time on `sigma`:

```
postProcess -func 'components(sigma)'
```

Components named `sigmaxx`, `sigmaxy` etc. are written to time directories of the case. The  $\sigma_{xx}$  stresses can be viewed in `paraFoam` as shown in Figure 5.4.

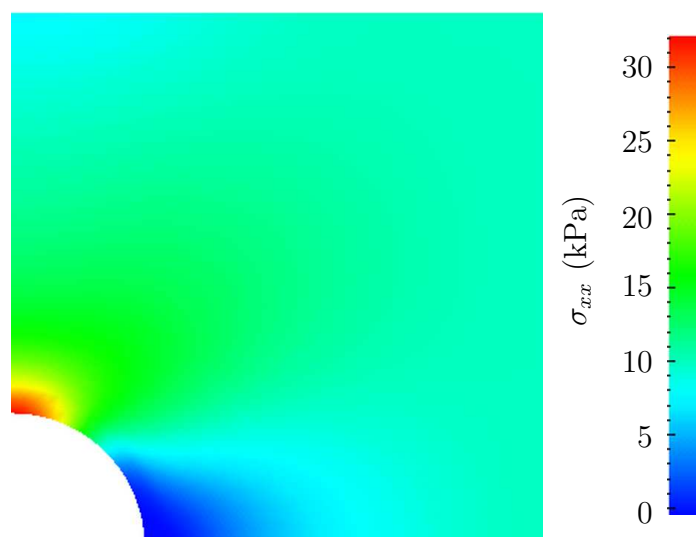
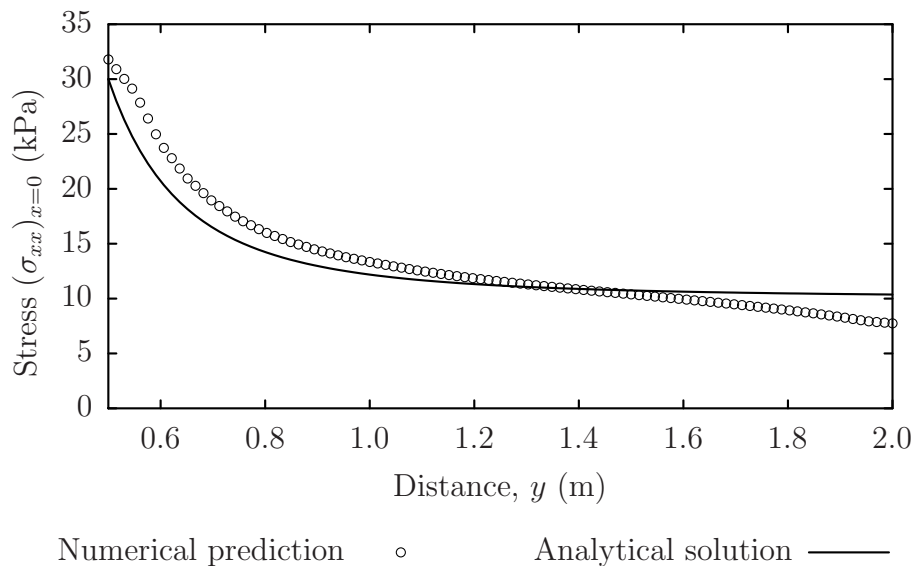


Figure 5.4:  $\sigma_{xx}$  stress field in the plate with hole.

We would like to compare the analytical solution of Equation 5.1 to our solution. We therefore must output a set of data of  $\sigma_{xx}$  along the left edge symmetry plane of our domain. The user may generate the required graph data using the `postProcess` utility, using a `sets` function object. The utility can be driven from a user-supplied file located in the `system` directory, whose entries are summarised in User Guide Table 7.3. The sample line specified in `sets` is set between (0.0, 0.5, 0.25) and (0.0, 2.0, 0.25), and the fields are specified in the `fields` list:

Figure 5.5: Normal stress along the vertical symmetry  $(\sigma_{xx})_{x=0}$ 

```

8
9  singleGraph
10 {
11     start    (0 0.5 0.25);
12     end      (0 2 0.25);
13     fields   (sigmaxx);
14
15     #includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"
16
17     setConfig
18     {
19         axis    y;
20     }
21
22     // Must be last entry
23     #includeEtc "caseDicts/postProcessing/graphs/graph.cfg"
24 }
25
26
27 // *****

```

The `writeFormat` is `raw 2` column format. The data is written into files within time subdirectories of a `sets` directory, *e.g.* the data at  $t = 100$  s is found within the file `sets/100/leftPatch_sigmaxx.xy`. In an application such as `GnuPlot`, one could type the following at the command prompt would be sufficient to plot both the numerical data and analytical solution:

```

plot [0.5:2] [0:] 'sets/100/leftPatch_sigmaxx.xy',
    1e4*(1+(0.125/(x**2)))+(0.09375/(x**4)))

```

An example plot is shown in Figure 5.5.

## 5.1.5 Exercises

The user may wish to experiment with `solidDisplacementFoam` by trying the following exercises:

### 5.1.5.1 Increasing mesh resolution

Increase the mesh resolution in each of the  $x$  and  $y$  directions. Use `mapFields` to map the final coarse mesh results from section 5.1.4 to the initial conditions for the fine mesh.

### 5.1.5.2 Introducing mesh grading

Grade the mesh so that the cells near the hole are finer than those away from the hole. Design the mesh so that the ratio of sizes between adjacent cells is no more than 1.1 and so that the ratio of cell sizes between blocks is similar to the ratios within blocks. Mesh grading is described in section 2.1.6. Again use `mapFields` to map the final coarse mesh results from section 5.1.4 to the initial conditions for the graded mesh. Compare the results with those from the analytical solution and previous calculations. Can this solution be improved upon using the same number of cells with a different solution?

### 5.1.5.3 Changing the plate size

The analytical solution is for an infinitely large plate with a finite sized hole in it. Therefore this solution is not completely accurate for a finite sized plate. To estimate the error, increase the plate size while maintaining the hole size at the same value.

# Index

**Symbols Numbers A B C D E F G H I J K L M N O P Q R S T U V W X Z**

## A

`adjustableRunTime`  
     keyword entry, [T-81](#)  
`adjustTimeStep` keyword, [T-81](#)  
`alphaContactAngle`  
     boundary condition, [T-78](#)  
 analytical solution, [T-45](#)  
 Annotation window panel, [T-22](#)  
 axes  
     right-handed rectangular Cartesian, [T-16](#)

## B

background  
     process, [T-23](#)  
`blockMesh` solver, [T-46](#)  
`blockMesh` utility, [T-34](#)  
`blockMeshDict`  
     dictionary, [T-16](#), [T-18](#), [T-33](#), [T-93](#)  
`blocks` keyword, [T-18](#), [T-28](#)  
 boundary condition  
     `alphaContactAngle`, [T-78](#)  
     empty, [T-16](#), [T-54](#), [T-69](#)  
     inlet, [T-54](#)  
     outlet, [T-54](#)  
     `pressureOutlet`, [T-69](#)  
     setup, [T-18](#)  
     `symmetryPlane`, [T-69](#)  
     wall, [T-37](#), [T-54](#), [T-69](#), [T-78](#)  
`boundaryField` keyword, [T-19](#)  
`boxToCell` keyword, [T-79](#)  
 breaking of a dam, [T-76](#)  
 button  
     Camera Parallel Projection, [T-22](#)  
     Enable Line Series, [T-32](#)  
     Orientation Axes, [T-22](#)  
     Refresh Times, [T-23](#)  
     Rescale to Data Range, [T-23](#)  
     Toggle Advanced Properties, [T-22](#)

## C

`cAlpha` keyword, [T-82](#)  
 Camera Parallel Projection button, [T-22](#)  
 cavity flow, [T-16](#)

Color Legend window, [T-25](#)  
`controlDict`  
     dictionary, [T-20](#), [T-29](#), [T-38](#), [T-71](#), [T-81](#), [T-95](#)  
`controlDict` file, [T-49](#)  
 convergence, [T-36](#)  
 coordinate system, [T-16](#)  
 Courant number, [T-20](#)  
`CrossPowerLaw`  
     keyword entry, [T-80](#)  
 Current Time Controls menu, [T-23](#)  
 cylinder  
     flow around a, [T-45](#)

## D

dam  
     breaking of a, [T-76](#)  
 decompression of a tank, [T-68](#)  
`defaultFieldValues` keyword, [T-79](#)  
 dictionary  
     *PISO*, [T-21](#)  
     `blockMeshDict`, [T-16](#), [T-18](#), [T-33](#), [T-93](#)  
     `controlDict`, [T-20](#), [T-29](#), [T-38](#), [T-71](#), [T-81](#), [T-95](#)  
     *fvSchemes*, [T-82](#)  
     *mechanicalProperties*, [T-95](#)  
     *thermalProperties*, [T-95](#)  
     *transportProperties*, [T-19](#), [T-35](#), [T-38](#)  
     *turbulenceProperties*, [T-38](#), [T-81](#)  
 dimensions keyword, [T-19](#)  
 directory  
     system, [T-49](#)  
     tutorials, [T-13](#)

## E

empty  
     boundary condition, [T-16](#), [T-54](#), [T-69](#)  
 Enable Line Series button, [T-32](#)  
`endTime` keyword, [T-20](#)

## F

field  
     U, [T-21](#)

p, [T-21](#)  
 fieldValues keyword, [T-79](#)  
 file  
   *controlDict*, [T-49](#)  
   *g*, [T-80](#)  
   *transportProperties*, [T-79](#)  
 flow  
   free surface, [T-76](#)  
   laminar, [T-16](#)  
   steady, turbulent, [T-58](#)  
   supersonic, [T-64](#)  
   turbulent, [T-16](#)  
 flow around a cylinder, [T-45](#)  
 flow over backward step, [T-58](#)  
 foreground  
   process, [T-23](#)  
*fvSchemes*  
   dictionary, [T-82](#)

## G

*g* file, [T-80](#)  
 GAMG  
   keyword entry, [T-97](#)  
 gradient  
   Gauss's theorem, [T-96](#)  
   least square fit, [T-96](#)  
   least squares method, [T-96](#)

## I

icoFoam solver, [T-16](#), [T-19](#), [T-20](#), [T-23](#)  
 incompressibleTransportModels  
   library, [T-59](#)  
 incompressibleTurbulenceModels  
   library, [T-59](#)  
 inlet  
   boundary condition, [T-54](#)  
 internalField keyword, [T-19](#)

## K

keyword  
   adjustTimeStep, [T-81](#)  
   blocks, [T-18](#), [T-28](#)  
   boundaryField, [T-19](#)  
   boxToCell, [T-79](#)  
   cAlpha, [T-82](#)  
   defaultFieldValues, [T-79](#)  
   dimensions, [T-19](#)  
   endTime, [T-20](#)  
   fieldValues, [T-79](#)  
   internalField, [T-19](#)  
   latestTime, [T-35](#)  
   leastSquares, [T-96](#)  
   maxAlphaCo, [T-81](#)  
   maxCo, [T-81](#)  
   maxDeltaT, [T-81](#)

nAlphaSubCycles, [T-82](#)  
 pRefCell, [T-21](#)  
 pRefValue, [T-21](#)  
 pressure, [T-95](#)  
 printCoeffs, [T-38](#)  
 regions, [T-79](#)  
 relTol, [T-97](#)  
 simulationType, [T-38](#), [T-81](#)  
 solver, [T-97](#)  
 startFrom, [T-20](#)  
 startTime, [T-20](#)  
 tolerance, [T-97](#)  
 topoSetSource, [T-79](#)  
 traction, [T-95](#)  
 value, [T-19](#)  
 vertices, [T-18](#)  
 writeControl, [T-20](#), [T-81](#)  
 writeFormat, [T-99](#)  
 writeInterval, [T-20](#), [T-29](#)

keyword entry  
   CrossPowerLaw, [T-80](#)  
   GAMG, [T-97](#)  
   LES, [T-38](#)  
   Newtonian, [T-80](#)  
   RAS, [T-38](#)  
   adjustableRunTime, [T-81](#)  
   laminar, [T-38](#)  
   runTime, [T-29](#)  
   startTime, [T-20](#)  
   timeStep, [T-20](#), [T-29](#)

## L

laminar  
   keyword entry, [T-38](#)  
 latestTime keyword, [T-35](#)  
 leastSquares keyword, [T-96](#)  
 LES  
   keyword entry, [T-38](#)  
 library  
   incompressibleTransportModels, [T-59](#)  
   incompressibleTurbulenceModels, [T-59](#)  
 lid-driven cavity flow, [T-16](#)  
 Line Style menu, [T-32](#)  
 liquid  
   electrically-conducting, [T-53](#)

## M

magnetohydrodynamics, [T-53](#)  
 mapFields utility, [T-28](#), [T-35](#), [T-39](#), [T-99](#)  
 Marker Style menu, [T-32](#)  
 maxAlphaCo keyword, [T-81](#)  
 maxCo keyword, [T-81](#)  
 maxDeltaT keyword, [T-81](#)  
*mechanicalProperties*

dictionary, [T-95](#)

menu

- Current Time Controls, [T-23](#)
- Line Style, [T-32](#)
- Marker Style, [T-32](#)
- VCR Controls, [T-23](#)

menu entry

- Plot Over Line, [T-32](#)
- Settings..., [T-22](#)
- Show Color Legend, [T-23](#)

mesh

- grading, example of, [T-58](#)
- non-orthogonal, [T-45](#)
- refinement, [T-68](#)
- resolution, [T-28](#)

Mesh Parts window panel, [T-22](#)

mhdFoam solver, [T-54](#)

## N

nAlphaSubCycles keyword, [T-82](#)

Newtonian

- keyword entry, [T-80](#)

non-orthogonal mesh, [T-45](#)

## O

Orientation Axes button, [T-22](#)

outlet

- boundary condition, [T-54](#)

## P

p field, [T-21](#)

paraFoam, [T-21](#)

Pipeline Browser window, [T-22](#)

*PISO*

- dictionary, [T-21](#)

pisoFoam solver, [T-16](#)

Plot Over Line

- menu entry, [T-32](#)

potentialFoam solver, [T-46](#)

pRefCell keyword, [T-21](#)

pRefValue keyword, [T-21](#)

pressure keyword, [T-95](#)

pressure waves

- in liquids, [T-68](#)

pressureOutlet

- boundary condition, [T-69](#)

printCoeffs keyword, [T-38](#)

process

- background, [T-23](#)
- foreground, [T-23](#)

Properties window panel, [T-22](#), [T-23](#)

## R

RAS

- keyword entry, [T-38](#)

Refresh Times button, [T-23](#)

regions keyword, [T-79](#)

relTol keyword, [T-97](#)

Rescale to Data Range button, [T-23](#)

restart, [T-35](#)

Reynolds number, [T-16](#), [T-19](#)

runTime

- keyword entry, [T-29](#)

## S

setFields utility, [T-79](#)

Settings...

- menu entry, [T-22](#)

Show Color Legend

- menu entry, [T-23](#)

simpleFoam solver, [T-59](#)

simulationType keyword, [T-38](#), [T-81](#)

solidDisplacementFoam solver, [T-95](#)

solver

- blockMesh, [T-46](#)
- icoFoam, [T-16](#), [T-19](#), [T-20](#), [T-23](#)
- mhdFoam, [T-54](#)
- pisoFoam, [T-16](#)
- potentialFoam, [T-46](#)
- simpleFoam, [T-59](#)
- solidDisplacementFoam, [T-95](#)
- sonicFoam, [T-65](#)
- sonicLiquidFoam, [T-69](#)

solver keyword, [T-97](#)

sonicFoam solver, [T-65](#)

sonicLiquidFoam solver, [T-69](#)

startFrom keyword, [T-20](#)

startTime

- keyword entry, [T-20](#)

startTime keyword, [T-20](#)

steady flow

- turbulent, [T-58](#)

stress analysis of plate with hole, [T-90](#)

supersonic flow, [T-64](#)

supersonic flow over forward step, [T-64](#)

symmetryPlane

- boundary condition, [T-69](#)

system directory, [T-49](#)

## T

*thermalProperties*

- dictionary, [T-95](#)

time step, [T-20](#)

timeStep

- keyword entry, [T-20](#), [T-29](#)

Toggle Advanced Properties button, [T-22](#)

tolerance keyword, [T-97](#)

topoSetSource keyword, [T-79](#)

traction keyword, [T-95](#)

*transportProperties*

dictionary, [T-19](#), [T-35](#), [T-38](#)

*transportProperties* file, [T-79](#)

## turbulence

dissipation, [T-36](#)

kinetic energy, [T-36](#)

length scale, [T-37](#)

## turbulence model

RAS, [T-36](#)

*turbulenceProperties*

dictionary, [T-38](#), [T-81](#)

## turbulent flow

steady, [T-58](#)

## tutorials

breaking of a dam, [T-76](#)

decompression of a tank, [T-68](#)

flow around a cylinder, [T-45](#)

flow over backward step, [T-58](#)

Hartmann problem, [T-53](#)

introduction, [T-13](#)

lid-driven cavity flow, [T-16](#)

stress analysis of plate with hole, [T-90](#)

supersonic flow over forward step, [T-64](#)

*tutorials* directory, [T-13](#)

## U

U field, [T-21](#)

Ucomponents utility, [T-55](#)

upwind differencing, [T-82](#)

## utility

Ucomponents, [T-55](#)

blockMesh, [T-34](#)

mapFields, [T-28](#), [T-35](#), [T-39](#), [T-99](#)

setFields, [T-79](#)

## V

value keyword, [T-19](#)

VCR Controls menu, [T-23](#)

vertices keyword, [T-18](#)

## viscosity

kinematic, [T-20](#), [T-38](#)

## W

## wall

boundary condition, [T-37](#), [T-54](#), [T-69](#),  
[T-78](#)

## window

*Color Legend*, [T-25](#)

*Pipeline Browser*, [T-22](#)

## window panel

*Annotation*, [T-22](#)

*Mesh Parts*, [T-22](#)

*Properties*, [T-22](#), [T-23](#)

writeControl keyword, [T-20](#), [T-81](#)

writeFormat keyword, [T-99](#)

writeInterval keyword, [T-20](#), [T-29](#)