

Copyright © 2000, 2001, 2002, 2003, 2004, 2005, 2006 OpenCFD Limited.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Back-Cover Texts and one Front-Cover Text: “Available free from openfoam.org.” A copy of the license is included in the section entitled “GNU Free Documentation License”.

This document is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Typeset in L^AT_EX.

Open  FOAM

The Open Source CFD Toolbox

Programmer's Guide

Version 1.3
28th March 2006

GNU Free Documentation License

Version 1.2, November 2002

Copyright ©2000,2001,2002 Free Software Foundation, Inc.

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the

document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of

any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Trademarks

ANSYS is a registered trademark of ANSYS Inc.

CFX is a registered trademark of AEA Technology Engineering Software Ltd.

CHEMKIN is a registered trademark of Sandia National Laboratories

CORBA is a registered trademark of Object Management Group Inc.

openDX is a registered trademark of International Business Machines Corporation

EnSight is a registered trademark of Computational Engineering International Ltd.

AVS/Express is a registered trademark of Advanced Visual Systems Inc.

Fluent is a registered trademark of Fluent Inc.

GAMBIT is a registered trademark of Fluent Inc.

Fieldview is a registered trademark of Intelligent Light

Icem-CFD is a registered trademark of ICEM Technologies GmbH

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

MICO is a registered trademark of MICO Inc.

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

| | | | |
|---|-------------|--|--|
| Copyright Notice | P-2 | | |
| GNU Free Documentation Licence | P-3 | | |
| 1. APPLICABILITY AND DEFINITIONS | P-3 | | |
| 2. VERBATIM COPYING | P-4 | | |
| 3. COPYING IN QUANTITY | P-4 | | |
| 4. MODIFICATIONS | P-5 | | |
| 5. COMBINING DOCUMENTS | P-6 | | |
| 6. COLLECTIONS OF DOCUMENTS | P-7 | | |
| 7. AGGREGATION WITH INDEPENDENT WORKS | P-7 | | |
| 8. TRANSLATION | P-7 | | |
| 9. TERMINATION | P-7 | | |
| 10. FUTURE REVISIONS OF THIS LICENSE | P-7 | | |
| Trademarks | P-9 | | |
| Contents | P-11 | | |
| 1 Tensor mathematics | P-15 | | |
| 1.1 Coordinate system | P-15 | | |
| 1.2 Tensors | P-15 | | |
| 1.2.1 Tensor notation | P-17 | | |
| 1.3 Algebraic tensor operations | P-17 | | |
| 1.3.1 The inner product | P-18 | | |
| 1.3.2 The double inner product of two tensors | P-19 | | |
| 1.3.3 The triple inner product of two third rank tensors | P-19 | | |
| 1.3.4 The outer product | P-19 | | |
| 1.3.5 The cross product of two vectors | P-19 | | |
| 1.3.6 Other general tensor operations | P-20 | | |
| 1.3.7 Geometric transformation and the identity tensor | P-20 | | |
| 1.3.8 Useful tensor identities | P-21 | | |
| 1.3.9 Operations exclusive to tensors of rank 2 | P-21 | | |
| 1.3.10 Operations exclusive to scalars | P-22 | | |
| 1.4 OpenFOAM tensor classes | P-23 | | |
| 1.4.1 Algebraic tensor operations in OpenFOAM | P-23 | | |
| 1.5 Dimensional units | P-25 | | |
| 2 Discretisation procedures | P-27 | | |
| 2.1 Differential operators | P-27 | | |
| 2.1.1 Gradient | P-27 | | |
| 2.1.2 Divergence | P-28 | | |
| 2.1.3 Curl | P-28 | | |
| 2.1.4 Laplacian | P-28 | | |
| 2.1.5 Temporal derivative | P-28 | | |
| 2.2 Overview of discretisation | P-29 | | |
| 2.2.1 OpenFOAM lists and fields | P-29 | | |
| 2.3 Discretisation of the solution domain | P-29 | | |
| 2.3.1 Defining a mesh in OpenFOAM | P-31 | | |
| 2.3.2 Defining a <code>geometricField</code> in OpenFOAM | P-32 | | |
| 2.4 Equation discretisation | P-33 | | |
| 2.4.1 The Laplacian term | P-38 | | |
| 2.4.2 The convection term | P-38 | | |
| 2.4.3 First time derivative | P-39 | | |
| 2.4.4 Second time derivative | P-39 | | |
| 2.4.5 Divergence | P-39 | | |
| 2.4.6 Gradient | P-40 | | |
| 2.4.7 Grad-grad squared | P-41 | | |
| 2.4.8 Curl | P-41 | | |
| 2.4.9 Source terms | P-41 | | |
| 2.4.10 Other explicit discretisation schemes | P-41 | | |
| 2.5 Temporal discretisation | P-42 | | |
| 2.5.1 Treatment of temporal discretisation in OpenFOAM | P-43 | | |
| 2.6 Boundary Conditions | P-43 | | |
| 2.6.1 Physical boundary conditions | P-44 | | |
| 3 Examples of the use of OpenFOAM | P-45 | | |
| 3.1 Flow around a cylinder | P-45 | | |
| 3.1.1 Problem specification | P-46 | | |
| 3.1.2 Note on <code>potentialFoam</code> | P-47 | | |
| 3.1.3 Mesh generation | P-47 | | |
| 3.1.4 Boundary conditions and initial fields | P-49 | | |
| 3.1.5 Running the case | P-49 | | |
| 3.1.6 Generating the analytical solution | P-50 | | |
| 3.1.7 Exercise | P-53 | | |
| 3.2 Steady turbulent flow over a backward-facing step | P-54 | | |
| 3.2.1 Problem specification | P-54 | | |
| 3.2.2 Mesh generation | P-55 | | |
| 3.2.3 Boundary conditions and initial fields | P-58 | | |
| 3.2.4 Case control | P-58 | | |
| 3.2.5 Running the case and post-processing | P-59 | | |
| 3.3 Supersonic flow over a forward-facing step | P-59 | | |
| 3.3.1 Problem specification | P-59 | | |
| 3.3.2 Mesh generation | P-61 | | |
| 3.3.3 Running the case | P-62 | | |
| 3.3.4 Exercise | P-62 | | |
| 3.4 Decompression of a tank internally pressurised with water | P-62 | | |
| 3.4.1 Problem specification | P-63 | | |
| 3.4.2 Mesh Generation | P-64 | | |
| 3.4.3 Preparing the Run | P-65 | | |
| 3.4.4 Running the case | P-66 | | |
| 3.4.5 Improving the solution by refining the mesh | P-67 | | |
| 3.5 Magnetohydrodynamic flow of a liquid | P-67 | | |

| | | |
|--------------|---------------------------------|-------------|
| 3.5.1 | Problem specification | P-68 |
| 3.5.2 | Mesh generation | P-70 |
| 3.5.3 | Running the case | P-71 |
| Index | | P-73 |

aligned with the right-handed rectangular Cartesian axes x , y and z respectively. The base vectors are therefore orthogonal, *i.e.* at right-angles to one another. Every tensor has the following attributes:

Dimension d of the particular space to which they belong, *i.e.* $d = 3$ in OpenFOAM;

Rank An integer $r \geq 0$, such that the number of component values = d^r .

While OpenFOAM 1.x is set to 3 dimensions, it offers tensors of ranks 0 to 3 as standard while being written in such a way to allow this basic set of ranks to be extended indefinitely. Tensors of rank 0 and 1, better known as scalars and vectors, should be familiar to readers; tensors of rank 2 and 3 may not be so familiar. For completeness all ranks of tensor offered as standard in OpenFOAM 1.x are reviewed below.

Rank 0 ‘scalar’ Any property which can be represented by a single real number, denoted by characters in italics, *e.g.* mass m , volume V , pressure p and viscosity μ .

Rank 1 ‘vector’ An entity which can be represented physically by both magnitude and direction. In component form, the vector $\mathbf{a} = (a_1, a_2, a_3)$ relates to a set of Cartesian axes x, y, z respectively. The *index notation* presents the same vector as a_i , $i = 1, 2, 3$, although the list of indices $i = 1, 2, 3$ will be omitted in this book, as it is intuitive since we are always dealing with 3 dimensions.

Rank 2 ‘tensor’ or second rank tensor, \mathbf{T} has 9 components which can be expressed in array notation as:

$$\mathbf{T} = T_{ij} = \begin{pmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{pmatrix} \quad (1.1)$$

The components T_{ij} are now represented using 2 indices since $r = 2$ and the list of indices $i, j = 1, 2, 3$ is omitted as before. The components for which $i = j$ are referred to as the diagonal components, and those for which $i \neq j$ are referred to as the off-diagonal components. The *transpose* of \mathbf{T} is produced by exchanging components across the diagonal such that

$$\mathbf{T}^T = T_{ji} = \begin{pmatrix} T_{11} & T_{21} & T_{31} \\ T_{12} & T_{22} & T_{32} \\ T_{13} & T_{23} & T_{33} \end{pmatrix} \quad (1.2)$$

Note: a rank 2 tensor is often colloquially termed ‘tensor’ since the occurrence of higher order tensors is fairly rare.

Symmetric rank 2 The term ‘symmetric’ refers to components being symmetric about the diagonal, *i.e.* $T_{ij} = T_{ji}$. In this case, there are only 6 independent components since $T_{12} = T_{21}$, $T_{13} = T_{31}$ and $T_{23} = T_{32}$. OpenFOAM distinguishes between symmetric and non-symmetric tensors to save memory by storing 6 components rather than 9 if the tensor is symmetric. Most tensors encountered in continuum mechanics are symmetric.

Rank 3 has 27 components and is represented in index notation as P_{ijk} which is too long to represent in array notation as in [Equation 1.1](#).

Symmetric rank 3 Symmetry of a rank 3 tensor is defined in OpenFOAM to mean that $P_{ijk} = P_{ikj} = P_{jik} = P_{jki} = P_{kij} = P_{kji}$ and therefore has 10 independent components. More specifically, it is formed by the outer product of 3 identical vectors, where the outer product operation is described in [Section 1.3.4](#).

Chapter 1

Tensor mathematics

This Chapter describes tensors and their algebraic operations and how they are represented in mathematical text in this book. It then explains how tensors and tensor algebra are programmed in OpenFOAM.

1.1 Coordinate system

OpenFOAM is primarily designed to solve problems in continuum mechanics, *i.e.* the branch of mechanics concerned with the stresses in solids, liquids and gases and the deformation or flow of these materials. OpenFOAM is therefore based in 3 dimensional space and time and deals with physical entities described by tensors. The coordinate system used by OpenFOAM is the right-handed rectangular Cartesian axes as shown in [Figure 1.1](#). This system of axes is constructed by defining an origin O from which three lines are drawn at right angles to each other, termed the Ox , Oy , Oz axes. A right-handed set of axes is defined such that to an observer looking down the Oz axis (with O nearest them), the arc from a point on the Ox axis to a point on the Oy axis is in a clockwise sense.

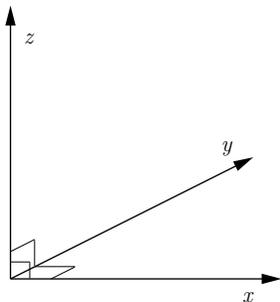


Figure 1.1: Right handed axes

1.2 Tensors

The term tensor describes an entity that belongs to a particular space and obeys certain mathematical rules. Briefly, tensors are represented by a set of *component values* relating to a set of unit base vectors; in OpenFOAM the unit base vectors \mathbf{i}_x , \mathbf{i}_y and \mathbf{i}_z are

1.2.1 Tensor notation

This is a book on computational continuum mechanics that deals with problems involving complex PDEs in 3 spatial dimensions and in time. It is vital from the beginning to adopt a notation for the equations which is compact yet unambiguous. To make the equations easy to follow, we must use a notation that encapsulates the idea of a tensor as an entity in its own right, rather than a list of scalar components. Additionally, any tensor operation should be perceived as an operation on the entire tensor entity rather than a series of operations on its components.

Consequently, in this book the *tensor notation* is preferred in which any tensor of rank 1 and above, *i.e.* all tensors other than scalars, are represented by letters in bold face, *e.g.* \mathbf{a} . This actively promotes the concept of a tensor as an entity in its own right since it is denoted by a single symbol, and it is also extremely compact. The potential drawback is that the rank of a bold face symbol is not immediately apparent, although it is clearly not zero. However, in practice this presents no real problem since we are aware of the property each symbol represents and therefore intuitively know its rank, *e.g.* we know velocity \mathbf{U} is a tensor of rank 1.

A further, more fundamental idea regarding the choice of notation is that the mathematical representation of a tensor should not change depending on our coordinate system, *i.e.* the vector \mathbf{a} is the same vector irrespective of where we view it from. The tensor notation supports this concept as it implies nothing about the coordinate system. However, other notations, *e.g.* a_i , expose the individual components of the tensor which naturally implies the choice of coordinate system. The unsatisfactory consequence of this is that the tensor is then represented by a set of values which are not unique — they depend on the coordinate system.

That said, the index notation, introduced in Section 1.2, is adopted from time to time in this book mainly to expand tensor operations into the constituent components. When using the index notation, we adopt the *summation convention* which states that whenever the same letter subscript occurs twice in a term, the that subscript is to be given all values, *i.e.* 1, 2, 3, and the results added together, *e.g.*

$$a_i b_i = \sum_{i=1}^3 a_i b_i = a_1 b_1 + a_2 b_2 + a_3 b_3 \quad (1.3)$$

In the remainder of the book the symbol \sum is omitted since the repeated subscript indicates the summation.

1.3 Algebraic tensor operations

This section describes all the algebraic operations for tensors that are available in OpenFOAM. Let us first review the most simple tensor operations: addition, subtraction, and scalar multiplication and division. Addition and subtraction are both commutative and associative and are only valid between tensors of the same rank. The operations are performed by addition/subtraction of respective components of the tensors, *e.g.* the subtraction of two vectors \mathbf{a} and \mathbf{b} is

$$\mathbf{a} - \mathbf{b} = a_i - b_i = (a_1 - b_1, a_2 - b_2, a_3 - b_3) \quad (1.4)$$

Multiplication of any tensor \mathbf{a} by a scalar s is also commutative and associative and is performed by multiplying all the tensor components by the scalar. For example,

$$s\mathbf{a} = sa_i = (sa_1, sa_2, sa_3) \quad (1.5)$$

Division between a tensor \mathbf{a} and a scalar is only relevant when the scalar is the second argument of the operation, *i.e.*

$$\mathbf{a}/s = a_i/s = (a_1/s, a_2/s, a_3/s) \quad (1.6)$$

Following these operations are a set of more complex products between tensors of rank 1 and above, described in the following Sections.

1.3.1 The inner product

The inner product operates on any two tensors of rank r_1 and r_2 such that the rank of the result $r = r_1 + r_2 - 2$. Inner product operations with tensors up to rank 3 are described below:

- The inner product of two vectors \mathbf{a} and \mathbf{b} is commutative and produces a scalar $s = \mathbf{a} \cdot \mathbf{b}$ where

$$s = a_i b_i = a_1 b_1 + a_2 b_2 + a_3 b_3 \quad (1.7)$$

- The inner product of a tensor \mathbf{T} and vector \mathbf{a} produces a vector $\mathbf{b} = \mathbf{T} \cdot \mathbf{a}$, represented below as a column array for convenience

$$b_i = T_{ij} a_j = \begin{pmatrix} T_{11}a_1 + T_{12}a_2 + T_{13}a_3 \\ T_{21}a_1 + T_{22}a_2 + T_{23}a_3 \\ T_{31}a_1 + T_{32}a_2 + T_{33}a_3 \end{pmatrix} \quad (1.8)$$

It is non-commutative if \mathbf{T} is non-symmetric such that $\mathbf{b} = \mathbf{a} \cdot \mathbf{T} = \mathbf{T}^T \cdot \mathbf{a}$ is

$$b_i = a_j T_{ji} = \begin{pmatrix} a_1 T_{11} + a_2 T_{21} + a_3 T_{31} \\ a_1 T_{12} + a_2 T_{22} + a_3 T_{32} \\ a_1 T_{13} + a_2 T_{23} + a_3 T_{33} \end{pmatrix} \quad (1.9)$$

- The inner product of two tensors \mathbf{T} and \mathbf{S} produces a tensor $\mathbf{P} = \mathbf{T} \cdot \mathbf{S}$ whose components are evaluated as:

$$P_{ij} = T_{ik} S_{kj} \quad (1.10)$$

It is non-commutative such that $\mathbf{T} \cdot \mathbf{S} = (\mathbf{S}^T \cdot \mathbf{T}^T)^T$

- The inner product of a vector \mathbf{a} and third rank tensor \mathbf{P} produces a second rank tensor $\mathbf{T} = \mathbf{a} \cdot \mathbf{P}$ whose components are

$$T_{ij} = a_k P_{kij} \quad (1.11)$$

Again this is non-commutative so that $\mathbf{T} = \mathbf{P} \cdot \mathbf{a}$ is

$$T_{ij} = P_{ijk} a_k \quad (1.12)$$

- The inner product of a second rank tensor \mathbf{T} and third rank tensor \mathbf{P} produces a third rank tensor $\mathbf{Q} = \mathbf{T} \cdot \mathbf{P}$ whose components are

$$Q_{ijk} = T_{il} P_{ljk} \quad (1.13)$$

Again this is non-commutative so that $\mathbf{Q} = \mathbf{P} \cdot \mathbf{T}$ is

$$Q_{ijk} = P_{ijl} T_{lk} \quad (1.14)$$

1.3.2 The double inner product of two tensors

The double inner product of two second-rank tensors \mathbf{T} and \mathbf{S} produces a scalar $s = \mathbf{T}:\mathbf{S}$ which can be evaluated as the sum of the 9 products of the tensor components

$$s = T_{ij}S_{ij} = T_{11}S_{11} + T_{12}S_{12} + T_{13}S_{13} + T_{21}S_{21} + T_{22}S_{22} + T_{23}S_{23} + T_{31}S_{31} + T_{32}S_{32} + T_{33}S_{33} \quad (1.15)$$

The double inner product between a second rank tensor \mathbf{T} and third rank tensor \mathbf{P} produces a vector $\mathbf{a} = \mathbf{T}:\mathbf{P}$ with components

$$a_i = T_{jk}P_{jki} \quad (1.16)$$

This is non-commutative so that $\mathbf{a} = \mathbf{P}:\mathbf{T}$ is

$$a_i = P_{ijk}T_{jk} \quad (1.17)$$

1.3.3 The triple inner product of two third rank tensors

The triple inner product of two third rank tensors \mathbf{P} and \mathbf{Q} produces a scalar $s = \mathbf{P}:\mathbf{Q}$ which can be evaluated as the sum of the 27 products of the tensor components

$$s = P_{ijk}Q_{ijk} \quad (1.18)$$

1.3.4 The outer product

The outer product operates between vectors and tensors as follows:

- The outer product of two vectors \mathbf{a} and \mathbf{b} is non-commutative and produces a tensor $\mathbf{T} = \mathbf{ab} = (\mathbf{ba})^T$ whose components are evaluated as:

$$T_{ij} = a_i b_j = \begin{pmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{pmatrix} \quad (1.19)$$

- An outer product of a vector \mathbf{a} and second rank tensor \mathbf{T} produces a third rank tensor $\mathbf{P} = \mathbf{aT}$ whose components are

$$P_{ijk} = a_i T_{jk} \quad (1.20)$$

This is non-commutative so that $\mathbf{P} = \mathbf{T}\mathbf{a}$ produces

$$P_{ijk} = T_{ij} a_k \quad (1.21)$$

1.3.5 The cross product of two vectors

The cross product operation is exclusive to vectors only. For two vectors \mathbf{a} with \mathbf{b} , it produces a vector $\mathbf{c} = \mathbf{a} \times \mathbf{b}$ whose components are

$$c_i = e_{ijk} a_j b_k = (a_2 b_3 - a_3 b_2, a_3 b_1 - a_1 b_3, a_1 b_2 - a_2 b_1) \quad (1.22)$$

where the *permutation symbol* is defined by

$$e_{ijk} = \begin{cases} 0 & \text{when any two indices are equal} \\ +1 & \text{when } i,j,k \text{ are an even permutation of } 1,2,3 \\ -1 & \text{when } i,j,k \text{ are an odd permutation of } 1,2,3 \end{cases} \quad (1.23)$$

in which the even permutations are 123, 231 and 312 and the odd permutations are 132, 213 and 321.

1.3.6 Other general tensor operations

Some less common tensor operations and terminology used by OpenFOAM are described below.

Square of a tensor is defined as the outer product of the tensor with itself, *e.g.* for a vector \mathbf{a} , the square $\mathbf{a}^2 = \mathbf{aa}$.

n th power of a tensor is evaluated by n outer products of the tensor, *e.g.* for a vector \mathbf{a} , the 3rd power $\mathbf{a}^3 = \mathbf{aaa}$.

Magnitude squared of a tensor is the r th inner product of the tensor of rank r with itself, to produce a scalar. For example, for a second rank tensor \mathbf{T} , $|\mathbf{T}|^2 = \mathbf{T}:\mathbf{T}$.

Magnitude is the square root of the magnitude squared, *e.g.* for a tensor \mathbf{T} , $|\mathbf{T}| = \sqrt{\mathbf{T}:\mathbf{T}}$. Vectors of unit magnitude are referred to as *unit vectors*.

Component maximum is the component of the tensor with greatest value, inclusive of sign, *i.e.* not the largest magnitude.

Component minimum is the component of the tensor with smallest value.

Component average is the mean of all components of a tensor.

Scale As the name suggests, the scale function is a tool for scaling the components of one tensor by the components of another tensor of the same rank. It is evaluated as the product of corresponding components of 2 tensors, *e.g.*, scaling vector \mathbf{a} by vector \mathbf{b} would produce vector \mathbf{c} whose components are

$$c_i = \text{scale}(\mathbf{a}, \mathbf{b}) = (a_1 b_1, a_2 b_2, a_3 b_3) \quad (1.24)$$

1.3.7 Geometric transformation and the identity tensor

A second rank tensor \mathbf{T} is strictly defined as a linear vector function, *i.e.* it is a function which associates an argument vector \mathbf{a} to another vector \mathbf{b} by the inner product $\mathbf{b} = \mathbf{T} \cdot \mathbf{a}$. The components of \mathbf{T} can be chosen to perform a specific geometric transformation of a tensor from the x, y, z coordinate system to a new coordinate system x^*, y^*, z^* ; \mathbf{T} is then referred to as the *transformation tensor*. While a scalar remains unchanged under a transformation, the vector \mathbf{a} is transformed to \mathbf{a}^* by

$$\mathbf{a}^* = \mathbf{T} \cdot \mathbf{a} \quad (1.25)$$

A second rank tensor \mathbf{S} is transformed to \mathbf{S}^* according to

$$\mathbf{S}^* = \mathbf{T} \cdot \mathbf{S} \cdot \mathbf{T}^T \quad (1.26)$$

The *identity tensor* \mathbf{I} is defined by the requirement that it transforms another tensor onto itself. For all vectors \mathbf{a}

$$\mathbf{a} = \mathbf{I} \cdot \mathbf{a} \quad (1.27)$$

and therefore

$$\mathbf{I} = \delta_{ij} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (1.28)$$

where δ_{ij} is known as the *Kronecker delta* symbol.

1.3.8 Useful tensor identities

Several identities are listed below which can be verified by under the assumption that all the relevant derivatives exist and are continuous. The identities are expressed for scalar s and vector \mathbf{a} .

$$\begin{aligned}
\nabla \cdot (\nabla \times \mathbf{a}) &\equiv 0 \\
\nabla \times (\nabla s) &\equiv \mathbf{0} \\
\nabla \cdot (s\mathbf{a}) &\equiv s\nabla \cdot \mathbf{a} + \mathbf{a} \cdot \nabla s \\
\nabla \times (s\mathbf{a}) &\equiv s\nabla \times \mathbf{a} + \nabla s \times \mathbf{a} \\
\nabla(\mathbf{a} \cdot \mathbf{b}) &\equiv \mathbf{a} \times (\nabla \times \mathbf{b}) + \mathbf{b} \times (\nabla \times \mathbf{a}) + (\mathbf{a} \cdot \nabla)\mathbf{b} + (\mathbf{b} \cdot \nabla)\mathbf{a} \\
\nabla \cdot (\mathbf{a} \times \mathbf{b}) &\equiv \mathbf{b} \cdot (\nabla \times \mathbf{a}) - \mathbf{a} \cdot (\nabla \times \mathbf{b}) \\
\nabla \times (\mathbf{a} \times \mathbf{b}) &\equiv \mathbf{a}(\nabla \cdot \mathbf{b}) - \mathbf{b}(\nabla \cdot \mathbf{a}) + (\mathbf{b} \cdot \nabla)\mathbf{a} - (\mathbf{a} \cdot \nabla)\mathbf{b} \\
\nabla \times (\nabla \times \mathbf{a}) &\equiv \nabla(\nabla \cdot \mathbf{a}) - \nabla^2 \mathbf{a} \\
(\nabla \times \mathbf{a}) \times \mathbf{a} &\equiv \mathbf{a} \cdot (\nabla \mathbf{a}) - \nabla(\mathbf{a} \cdot \mathbf{a})
\end{aligned} \tag{1.29}$$

It is sometimes useful to know the $\epsilon - \delta$ identity to help to manipulate equations in index notation:

$$e_{ijk}e_{irs} = \delta_{jr}\delta_{ks} - \delta_{js}\delta_{kr} \tag{1.30}$$

1.3.9 Operations exclusive to tensors of rank 2

There are several operations that manipulate the components of tensors of rank 2 that are listed below:

Transpose of a tensor $\mathbf{T} = T_{ij}$ is $\mathbf{T}^T = T_{ji}$ as described in [Equation 1.2](#).

Symmetric and skew (antisymmetric) tensors As discussed in [section 1.2](#), a tensor is said to be symmetric if its components are symmetric about the diagonal, i.e. $\mathbf{T} = \mathbf{T}^T$. A skew or antisymmetric tensor has $\mathbf{T} = -\mathbf{T}^T$ which intuitively implies that $T_{11} = T_{22} = T_{33} = 0$. Every second order tensor can be decomposed into symmetric and skew parts by

$$\mathbf{T} = \underbrace{\frac{1}{2}(\mathbf{T} + \mathbf{T}^T)}_{\text{symmetric}} + \underbrace{\frac{1}{2}(\mathbf{T} - \mathbf{T}^T)}_{\text{skew}} = \text{symm } \mathbf{T} + \text{skew } \mathbf{T} \tag{1.31}$$

Trace The trace of a tensor \mathbf{T} is a scalar, evaluated by summing the diagonal components

$$\text{tr } \mathbf{T} = T_{11} + T_{22} + T_{33} \tag{1.32}$$

Diagonal returns a vector whose components are the diagonal components of the second rank tensor \mathbf{T}

$$\text{diag } \mathbf{T} = (T_{11}, T_{22}, T_{33}) \tag{1.33}$$

Deviatoric and hydrostatic tensors Every second rank tensor \mathbf{T} can be decomposed into a deviatoric component, for which $\text{tr } \mathbf{T} = 0$ and a hydrostatic component of the form $\mathbf{T} = s\mathbf{I}$ where s is a scalar. Every second rank tensor can be decomposed into deviatoric and hydrostatic parts as follows:

$$\mathbf{T} = \underbrace{\mathbf{T} - \frac{1}{3}(\text{tr } \mathbf{T})\mathbf{I}}_{\text{deviatoric}} + \underbrace{\frac{1}{3}(\text{tr } \mathbf{T})\mathbf{I}}_{\text{hydrostatic}} = \text{dev } \mathbf{T} + \text{hyd } \mathbf{T} \tag{1.34}$$

Determinant The determinant of a second rank tensor is evaluated by

$$\begin{aligned}
\det \mathbf{T} &= \begin{vmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{vmatrix} = T_{11}(T_{22}T_{33} - T_{23}T_{32}) - \\
&\quad T_{12}(T_{21}T_{33} - T_{23}T_{31}) + \\
&\quad T_{13}(T_{21}T_{32} - T_{22}T_{31}) \\
&= \frac{1}{6}e_{ijk}e_{pqr}T_{ip}T_{jq}T_{kr}
\end{aligned} \tag{1.35}$$

Cofactors The *minors* of a tensor are evaluated for each component by deleting the row and column in which the component is situated and evaluating the resulting entries as a 2×2 *determinant*. For example, the minor of T_{12} is

$$\begin{vmatrix} T_{11} & T_{13} \\ T_{21} & T_{23} \\ T_{31} & T_{33} \end{vmatrix} = \begin{vmatrix} T_{21} & T_{23} \\ T_{31} & T_{33} \end{vmatrix} = T_{21}T_{33} - T_{23}T_{31} \tag{1.36}$$

The cofactors are *signed minors* where each minor is component is given a sign based on the rule

$$\begin{aligned}
&+ve \text{ if } i + j \text{ is even} \\
&-ve \text{ if } i + j \text{ is odd}
\end{aligned} \tag{1.37}$$

The cofactors of \mathbf{T} can be evaluated as

$$\text{cof } \mathbf{T} = \frac{1}{2}e_{jkr}e_{ist}T_{sk}T_{tr} \tag{1.38}$$

Inverse The inverse of a tensor can be evaluated as

$$\text{inv } \mathbf{T} = \frac{\text{cof } \mathbf{T}^T}{\det \mathbf{T}} \tag{1.39}$$

Hodge dual of a tensor is a vector whose components are

$$*\mathbf{T} = (T_{23}, -T_{13}, T_{12}) \tag{1.40}$$

1.3.10 Operations exclusive to scalars

OpenFOAM supports most of the well known functions that operate on scalars, *e.g.* square root, exponential, logarithm, sine, cosine *etc.*, a list of which can be found in [Table 1.2](#). There are 3 additional functions defined within OpenFOAM that are described below:

Sign of a scalar s is

$$\text{sgn}(s) = \begin{cases} 1 & \text{if } s \geq 0, \\ -1 & \text{if } s < 0. \end{cases} \tag{1.41}$$

Positive of a scalar s is

$$\text{pos}(s) = \begin{cases} 1 & \text{if } s \geq 0, \\ 0 & \text{if } s < 0. \end{cases} \tag{1.42}$$

Limit of a scalar s by the scalar n

$$\text{limit}(s, n) = \begin{cases} s & \text{if } s < n, \\ 0 & \text{if } s \geq n. \end{cases} \tag{1.43}$$

1.4 OpenFOAM tensor classes

OpenFOAM contains a C++ class library `primitive` that contains the classes for the tensor mathematics described so far. The basic tensor classes that are available as standard in OpenFOAM are listed in Table 1.1. The Table also lists the functions that allow the user to access individual components of a tensor, known as access functions.

| Rank | Common name | Basic class | Access functions |
|------|-------------|---------------------|---|
| 0 | Scalar | <code>scalar</code> | |
| 1 | Vector | <code>vector</code> | <code>x()</code> , <code>y()</code> , <code>z()</code> |
| 2 | Tensor | <code>tensor</code> | <code>xx()</code> , <code>xy()</code> , <code>xz()</code> ... |

Table 1.1: Basic tensor classes in OpenFOAM

We can declare the tensor

$$\mathbf{T} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad (1.44)$$

in OpenFOAM by the line:

```
tensor T(1, 2, 3, 4, 5, 6, 7, 8, 9);
```

We can then access the component T_{13} , or T_{zz} using the `xz()` access function. For instance the code

```
Info << "Txz = " << T.xz() << endl;
```

outputs to the screen:

```
Txz = 3
```

1.4.1 Algebraic tensor operations in OpenFOAM

The algebraic operations described in Section 1.3 are all available to the OpenFOAM tensor classes using syntax which closely mimics the notation used in written mathematics. Some functions are represented solely by descriptive functions, *e.g.* `symm()`, but others can also be executed using symbolic operators, *e.g.* `*`. All functions are listed in Table 1.2.

| Operation | Comment | Mathematical Description | Description in OpenFOAM |
|-----------------------|--------------------------------------|--------------------------------|-----------------------------|
| Addition | | $\mathbf{a} + \mathbf{b}$ | <code>a + b</code> |
| Subtraction | | $\mathbf{a} - \mathbf{b}$ | <code>a - b</code> |
| Scalar multiplication | | $s\mathbf{a}$ | <code>s * a</code> |
| Scalar division | | \mathbf{a}/s | <code>a / s</code> |
| Outer product | rank $\mathbf{a}, \mathbf{b} \geq 1$ | $\mathbf{a}\mathbf{b}$ | <code>a * b</code> |
| Inner product | rank $\mathbf{a}, \mathbf{b} \geq 1$ | $\mathbf{a} \cdot \mathbf{b}$ | <code>a & b</code> |
| Double inner product | rank $\mathbf{a}, \mathbf{b} \geq 2$ | $\mathbf{a} : \mathbf{b}$ | <code>a && b</code> |
| Cross product | rank $\mathbf{a}, \mathbf{b} = 1$ | $\mathbf{a} \times \mathbf{b}$ | <code>a ^ b</code> |
| Square | | \mathbf{a}^2 | <code>sqr(a)</code> |

Continued on next page

Continued from previous page

| Operation | Comment | Mathematical Description | Description in OpenFOAM |
|--------------------------|---|--|-----------------------------|
| Magnitude squared | | $ \mathbf{a} ^2$ | <code>magSqr(a)</code> |
| Magnitude | | $ \mathbf{a} $ | <code>mag(a)</code> |
| Power | $n = 0, 1, \dots, 4$ | \mathbf{a}^n | <code>pow(a,n)</code> |
| Component average | $i = 1, \dots, N$ | \bar{a}_i | <code>cmptAv(a)</code> |
| Component maximum | $i = 1, \dots, N$ | $\max(a_i)$ | <code>max(a)</code> |
| Component minimum | $i = 1, \dots, N$ | $\min(a_i)$ | <code>min(a)</code> |
| Scale | | $\text{scale}(\mathbf{a}, \mathbf{b})$ | <code>scale(a,b)</code> |
| Geometric transformation | transforms \mathbf{a} using tensor \mathbf{T} | | <code>transform(T,a)</code> |

Operations exclusive to tensors of rank 2

| Operation | Comment | Mathematical Description | Description in OpenFOAM |
|--------------------------|---------|---------------------------|-------------------------|
| Transpose | | \mathbf{T}^T | <code>T.T()</code> |
| Diagonal | | $\text{diag } \mathbf{T}$ | <code>diag(T)</code> |
| Trace | | $\text{tr } \mathbf{T}$ | <code>tr(T)</code> |
| Deviatoric component | | $\text{dev } \mathbf{T}$ | <code>dev(T)</code> |
| Symmetric component | | $\text{symm } \mathbf{T}$ | <code>symm(T)</code> |
| Skew-symmetric component | | $\text{skew } \mathbf{T}$ | <code>skew(T)</code> |
| Determinant | | $\det \mathbf{T}$ | <code>det(T)</code> |
| Cofactors | | $\text{cof } \mathbf{T}$ | <code>cof(T)</code> |
| Inverse | | $\text{inv } \mathbf{T}$ | <code>inv(T)</code> |
| Hodge dual | | $*\mathbf{T}$ | <code>*T</code> |

Operations exclusive to scalars

| Operation | Comment | Mathematical Description | Description in OpenFOAM |
|-----------------------------------|------------|--------------------------|-------------------------|
| Sign (boolean) | | $\text{sgn}(s)$ | <code>sign(s)</code> |
| Positive (boolean) | | $s \geq 0$ | <code>pos(s)</code> |
| Negative (boolean) | | $s \leq 0$ | <code>neg(s)</code> |
| Limit | n scalar | $\text{limit}(s, n)$ | <code>limit(s,n)</code> |
| Square root | | \sqrt{s} | <code>sqr(s)</code> |
| Exponential | | $\exp s$ | <code>exp(s)</code> |
| Natural logarithm | | $\ln s$ | <code>log(s)</code> |
| Base 10 logarithm | | $\log_{10} s$ | <code>log10(s)</code> |
| Sine | | $\sin s$ | <code>sin(s)</code> |
| Cosine | | $\cos s$ | <code>cos(s)</code> |
| Tangent | | $\tan s$ | <code>tan(s)</code> |
| Arc sine | | $\text{asin } s$ | <code>asin(s)</code> |
| Arc cosine | | $\text{acos } s$ | <code>acos(s)</code> |
| Arc tangent | | $\text{atan } s$ | <code>atan(s)</code> |
| Hyperbolic sine | | $\sinh s$ | <code>sinh(s)</code> |
| Hyperbolic cosine | | $\cosh s$ | <code>cosh(s)</code> |
| Hyperbolic tangent | | $\tanh s$ | <code>tanh(s)</code> |
| Hyperbolic arc sine | | $\text{asinh } s$ | <code>asinh(s)</code> |
| Hyperbolic arc cosine | | $\text{acosh } s$ | <code>acosh(s)</code> |
| Hyperbolic arc tangent | | $\text{atanh } s$ | <code>atanh(s)</code> |
| Error function | | $\text{erf } s$ | <code>erf(s)</code> |
| Complement error function | | $\text{erfc } s$ | <code>erfc(s)</code> |
| Logarithm gamma function | | $\ln \Gamma s$ | <code>lgamma(s)</code> |
| Type 1 Bessel function of order 0 | | $J_0 s$ | <code>j0(s)</code> |
| Type 1 Bessel function of order 1 | | $J_1 s$ | <code>j1(s)</code> |

Continued on next page

Continued from previous page

| Operation | Comment | Mathematical Description | Description in OpenFOAM |
|-----------------------------------|---------|--------------------------|-------------------------|
| Type 2 Bessel function of order 0 | | $Y_0 s$ | <code>y0(s)</code> |
| Type 2 Bessel function of order 1 | | $Y_1 s$ | <code>y1(s)</code> |

\mathbf{a}, \mathbf{b} are tensors of arbitrary rank unless otherwise stated

s is a scalar, N is the number of tensor components

Table 1.2: Algebraic tensor operations in OpenFOAM

1.5 Dimensional units

In continuum mechanics, properties are represented in some chosen units, *e.g.* mass in kilograms (kg), volume in cubic metres (m^3), pressure in Pascals (kg m s^{-2}). Algebraic operations must be performed on these properties using consistent units of measurement; in particular, addition, subtraction and equality are only physically meaningful for properties of the same dimensional units. As a safeguard against implementing a meaningless operation, OpenFOAM encourages the user to attach dimensional units to any tensor and will then perform dimension checking of any tensor operation.

Units are defined using the `dimensionSet` class, *e.g.*

```
dimensionSet pressureDims(1, -1, -2, 0, 0, 0, 0);
```

| No. | Property | Unit | Symbol |
|-----|--------------------|----------|--------|
| 1 | Mass | kilogram | k |
| 2 | Length | metre | m |
| 3 | Time | second | s |
| 4 | Temperature | Kelvin | K |
| 5 | Quantity | moles | mol |
| 6 | Current | ampere | A |
| 7 | Luminous intensity | candela | cd |

Table 1.3: S.I. base units of measurement

where each of the values corresponds to the power of each of the S.I. base units of measurement listed in Table 1.3. The line of code declares `pressureDims` to be the `dimensionSet` for pressure kg m s^{-2} since the first entry in the `pressureDims` array, 1, corresponds to k^1 , the second entry, -1, corresponds to m^{-1} *etc.*. A tensor with units is defined using the `dimensioned<Type>` template class, the `<Type>` being `scalar`, `vector`, `tensor`, *etc.*. The `dimensioned<Type>` stores a variable name of class `word`, the value `<Type>` and a `dimensionSet`

```
dimensionedTensor sigma
(
    "sigma",
    dimensionSet(1, -1, -2, 0, 0, 0, 0),
    tensor(1e6,0,0,0,1e6,0,0,0,1e6),
);
```

creates a tensor with correct dimensions of pressure, or stress

$$\boldsymbol{\sigma} = \begin{pmatrix} 10^6 & 0 & 0 \\ 0 & 10^6 & 0 \\ 0 & 0 & 10^6 \end{pmatrix} \quad (1.45)$$

Chapter 2

Discretisation procedures

So far we have dealt with algebra of tensors at a point. The PDEs we wish to solve involve derivatives of tensors with respect to time and space. We therefore need to extend our description to a *tensor field*, *i.e.* a tensor that varies across time and spatial domains. In this Chapter we will first present a mathematical description of all the differential operators we may encounter. We will then show how a tensor field is constructed in OpenFOAM and how the derivatives of these fields are discretised into a set of algebraic equations.

2.1 Differential operators

Before defining the spatial derivatives we first introduce the nabla *vector operator* ∇ , represented in index notation as ∂_i :

$$\nabla \equiv \partial_i \equiv \frac{\partial}{\partial x_i} \equiv \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \frac{\partial}{\partial x_3} \right) \quad (2.1)$$

The nabla operator is a useful notation that obeys the following rules:

- it operates on the tensors to its right and the conventional rules of a derivative of a product, *e.g.* $\partial_i(ab) = (\partial_i a)b + a(\partial_i b)$;
- otherwise the nabla operator behaves like any other vector in an algebraic operation.

2.1.1 Gradient

If a scalar field s is defined and continuously differentiable then the gradient of s , ∇s is a vector field

$$\nabla s = \partial_i s = \left(\frac{\partial s}{\partial x_1}, \frac{\partial s}{\partial x_2}, \frac{\partial s}{\partial x_3} \right) \quad (2.2)$$

The gradient can operate on any tensor field to produce a tensor field that is one rank higher. For example, the gradient of a vector field \mathbf{a} is a second rank tensor field

$$\nabla \mathbf{a} = \partial_i a_j = \begin{pmatrix} \partial a_1/\partial x_1 & \partial a_2/\partial x_1 & \partial a_3/\partial x_1 \\ \partial a_1/\partial x_2 & \partial a_2/\partial x_2 & \partial a_3/\partial x_2 \\ \partial a_1/\partial x_3 & \partial a_2/\partial x_3 & \partial a_3/\partial x_3 \end{pmatrix} \quad (2.3)$$

2.1.2 Divergence

If a vector field \mathbf{a} is defined and continuously differentiable then the divergence of \mathbf{a} is a scalar field

$$\nabla \cdot \mathbf{a} = \partial_i a_i = \frac{\partial a_1}{\partial x_1} + \frac{\partial a_2}{\partial x_2} + \frac{\partial a_3}{\partial x_3} \quad (2.4)$$

The divergence can operate on any tensor field of rank 1 and above to produce a tensor that is one rank lower. For example the divergence of a second rank tensor field \mathbf{T} is a vector field (expanding the vector as a column array for convenience)

$$\nabla \cdot \mathbf{T} = \partial_i T_{ij} = \begin{pmatrix} \partial T_{11}/\partial x_1 + \partial T_{12}/\partial x_2 + \partial T_{13}/\partial x_3 \\ \partial T_{21}/\partial x_1 + \partial T_{22}/\partial x_2 + \partial T_{23}/\partial x_3 \\ \partial T_{31}/\partial x_1 + \partial T_{32}/\partial x_2 + \partial T_{33}/\partial x_3 \end{pmatrix} \quad (2.5)$$

2.1.3 Curl

If a vector field \mathbf{a} is defined and continuously differentiable then the curl of \mathbf{a} , $\nabla \times \mathbf{a}$ is a vector field

$$\nabla \times \mathbf{a} = e_{ijk} \partial_j a_k = \left(\frac{\partial a_3}{\partial x_2} - \frac{\partial a_2}{\partial x_3}, \frac{\partial a_1}{\partial x_3} - \frac{\partial a_3}{\partial x_1}, \frac{\partial a_2}{\partial x_1} - \frac{\partial a_1}{\partial x_2} \right) \quad (2.6)$$

The curl is related to the gradient by

$$\nabla \times \mathbf{a} = 2 (* \text{skew } \nabla \mathbf{a}) \quad (2.7)$$

2.1.4 Laplacian

The Laplacian is an operation that can be defined mathematically by a combination of the divergence and gradient operators by $\nabla^2 \equiv \nabla \cdot \nabla$. However, the Laplacian should be considered as a single operation that transforms a tensor field into another tensor field of the same rank, rather than a combination of two operations, one which raises the rank by 1 and one which reduces the rank by 1.

In fact, the Laplacian is best defined as a *scalar operator*, just as we defined nabla as a vector operator, by

$$\nabla^2 \equiv \partial^2 \equiv \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \frac{\partial^2}{\partial x_3^2} \quad (2.8)$$

For example, the Laplacian of a scalar field s is the scalar field

$$\nabla^2 s = \partial^2 s = \frac{\partial^2 s}{\partial x_1^2} + \frac{\partial^2 s}{\partial x_2^2} + \frac{\partial^2 s}{\partial x_3^2} \quad (2.9)$$

2.1.5 Temporal derivative

There is more than one definition of temporal, or time, derivative of a tensor. To describe the temporal derivatives we must first recall that the tensor relates to a property of a volume of material that may be moving. If we track an infinitesimally small volume of material, or particle, as it moves and observe the change in the tensorial property ϕ in time, we have the *total*, or *material* time derivative denoted by

$$\frac{D\phi}{Dt} = \lim_{\Delta t \rightarrow 0} \frac{\Delta\phi}{\Delta t} \quad (2.10)$$

However in continuum mechanics, particularly fluid mechanics, we often observe the change of a ϕ in time at a fixed point in space as different particles move across that point. This change at a point in space is termed the *spatial* time derivative which is denoted by $\partial/\partial t$ and is related to the material derivative by:

$$\frac{D\phi}{Dt} = \frac{\partial\phi}{\partial t} + \mathbf{U} \cdot \nabla\phi \quad (2.11)$$

where \mathbf{U} is the velocity field of property ϕ . The second term on the right is known as the convective rate of change of ϕ .

2.2 Overview of discretisation

The term discretisation means *approximation of a problem into discrete quantities*. The FV method and others, such as the finite element and finite difference methods, all discretise the problem as follows:

Spatial discretisation Defining the solution domain by a set of points that fill and bound a region of space when connected;

Temporal discretisation (For transient problems) dividing the time domain into into a finite number of time intervals, or steps;

Equation discretisation Generating a system of algebraic equations in terms of discrete quantities defined at specific locations in the domain, from the PDEs that characterise the problem.

2.2.1 OpenFOAM lists and fields

OpenFOAM frequently needs to store sets of data and perform functions, such as mathematical operations, on the data. OpenFOAM therefore provides an array template class `List<Type>`, making it possible to create a list of any object of class `Type` that inherits the functions of the `Type`. For example a List of vector is `List<vector>`.

Lists of the tensor classes are defined as standard in OpenFOAM by the template class `Field<Type>`. For better code legibility, all instances of `Field<Type>`, *e.g.* `Field<vector>`, are renamed using `typedef` declarations as `scalarField`, `vectorField`, `tensorField`, `symmTensorField`, `tensorThirdField` and `symmTensorThirdField`. Algebraic operations can be performed between `Fields` subject to obvious restrictions such as the fields having the same number of elements. OpenFOAM also supports operations between a field and single tensor, *e.g.* all values of a `Field U` can be multiplied by the scalar 2 with the operation `U = 2.0 * U`.

2.3 Discretisation of the solution domain

Discretisation of the solution domain is shown in [Figure 2.1](#). The space domain is discretised into computational mesh on which the PDEs are subsequently discretised. Discretisation of time, if required, is simple: it is broken into a set of time steps Δt that may change during a numerical simulation, perhaps depending on some condition calculated during the simulation.

On a more detailed level, discretisation of space requires the subdivision of the domain into a number of cells, or control volumes. The cells are contiguous, *i.e.* they do not

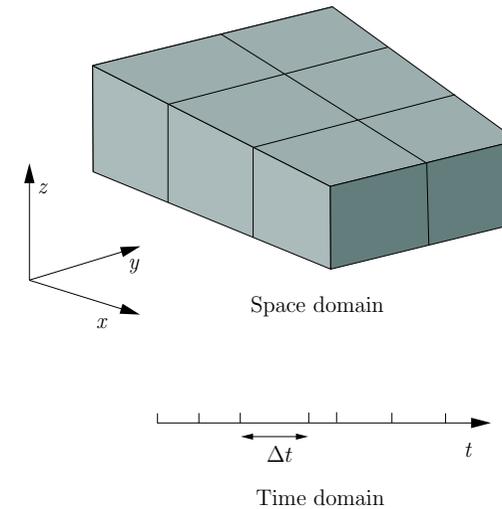


Figure 2.1: Discretisation of the solution domain

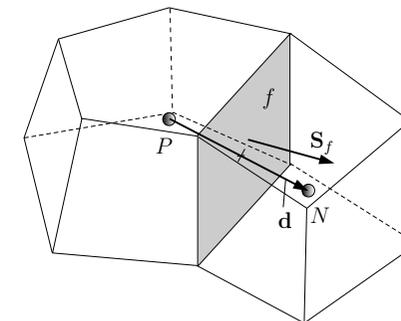


Figure 2.2: Parameters in finite volume discretisation

overlap one another and completely fill the domain. A typical cell is shown in [Figure 2.2](#). Dependent variables and other properties are principally stored at the cell centroid P although they may be stored on faces or vertices. The cell is bounded by a set of flat faces, given the generic label f . In OpenFOAM there is no limitation on the number of faces bounding each cell, nor any restriction on the alignment of each face. This kind of mesh is often referred to as “arbitrarily unstructured” to differentiate it from meshes in which the cell faces have a prescribed alignment, typically with the coordinate axes. Codes with arbitrarily unstructured meshes offer greater freedom in mesh generation and manipulation in particular when the geometry of the domain is complex or changes over time.

Whilst most properties are defined at the cell centroids, some are defined at cell faces. There are two types of cell face.

Internal faces Those faces that connect two cells (and it can never be more than two).

For each internal face, OpenFOAM designates one adjoining cell to be the face *owner* and the other to be the *neighbour*;

Boundary faces Those belonging to one cell since they coincide with the boundary of the domain. These faces simply have an owner cell.

2.3.1 Defining a mesh in OpenFOAM

There are different levels of mesh description in OpenFOAM, beginning with the most basic mesh class, named `polyMesh` since it is based on polyhedra. A `polyMesh` is constructed using the minimum information required to define the mesh geometry described below and presented in [Figure 2.3](#):

Points A list of cell vertex point coordinate vectors, *i.e.* a `vectorField`, that is renamed `pointField` using a `typedef` declaration;

Faces A list of cell faces `List<face>`, or `faceList`, where the `face` class is defined by a list of vertex numbers, corresponding to the `pointField`;

Cells a list of cells `List<cell>`, or `cellList`, where the `cell` class is defined by a list of face numbers, corresponding to the `faceList` described previously.

Boundary a `polyBoundaryMesh` decomposed into a list of patches, `polyPatchList` representing different regions of the boundary. The boundary is subdivided in this manner to allow different boundary conditions to be specified on different patches during a solution. All the faces of any `polyPatch` are stored as a single block of the `faceList`, so that its faces can be easily accessed using the `slice` class which stores references to the first and last face of the block. Each `polyPatch` is then constructed from

- a `slice`;
- a `word` to assign it a name.

FV discretisation uses specific data that is derived from the mesh geometry stored in `polyMesh`. OpenFOAM therefore extends the `polyMesh` class to `fvMesh` which stores the additional data needed for FV discretisation. `fvMesh` is constructed from `polyMesh` and stores the data in [Table 2.1](#) which can be updated during runtime in cases where the mesh moves, is refined *etc.*

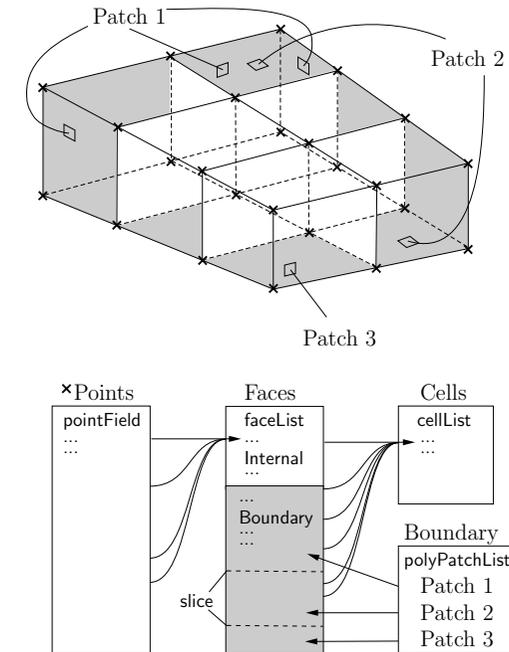


Figure 2.3: Schematic of the basic mesh description used in OpenFOAM

2.3.2 Defining a geometricField in OpenFOAM

So far we can define a field, *i.e.* a list of tensors, and a mesh. These can be combined to define a tensor field relating to discrete points in our domain, specified in OpenFOAM by the template class `geometricField<Type>`. The `Field` values are separated into those defined within the internal region of the domain, *e.g.* at the cell centres, and those defined on the domain boundary, *e.g.* on the boundary faces. The `geometricField<Type>` stores the following information:

Internal field This is simply a `Field<Type>`, described in [Section 2.2.1](#);

BoundaryField This is a `GeometricBoundaryField`, in which a `Field` is defined for the faces of each patch and a `Field` is defined for the patches of the boundary. This is then a field of fields, stored within an object of the `FieldField<Type>` class. A reference to the `fvBoundaryMesh` is also stored `[**]`.

Mesh A reference to an `fvMesh`, with some additional detail as to the whether the field is defined at cell centres, faces, *etc.*

Dimensions A `dimensionSet`, described in [Section 1.5](#).

Old values Discretisation of time derivatives requires field data from previous time steps. The `geometricField<Type>` will store references to stored fields from the previous, or old, time step and its previous, or old-old, time step where necessary.

| Class | Description | Symbol | Access function |
|--------------------|-----------------------|------------------|------------------|
| volScalarField | Cell volumes | V | $V()$ |
| surfaceVectorField | Face area vectors | \mathbf{S}_f | $\mathbf{Sf}()$ |
| surfaceScalarField | Face area magnitudes | $ \mathbf{S}_f $ | $\text{magSf}()$ |
| volVectorField | Cell centres | \mathbf{C} | $\mathbf{C}()$ |
| surfaceVectorField | Face centres | \mathbf{C}_f | $\mathbf{Cf}()$ |
| surfaceScalarField | Face motion fluxes ** | ϕ_g | $\text{phi}()$ |

Table 2.1: fvMesh stored data.

Previous iteration values The iterative solution procedures can use under-relaxation which requires access to data from the previous iteration. Again, if required, `geometricField<Type>` stores a reference to the data from the previous iteration.

As discussed in Section 2.3, we principally define a property at the cell centres but quite often it is stored at the cell faces and on occasion it is defined on cell vertices. The `geometricField<Type>` is renamed using `typedef` declarations to indicate where the field variable is defined as follows:

`volField<Type>` A field defined at cell centres;

`surfaceField<Type>` A field defined on cell faces;

`pointField<Type>` A field defined on cell vertices.

These `typedef` field classes of `geometricField<Type>` are illustrated in Figure 2.4. A `geometricField<Type>` inherits all the tensor algebra of `Field<Type>` and has all operations subjected to dimension checking using the `dimensionSet`. It can also be subjected to the FV discretisation procedures described in the following Section. The class structure used to build `geometricField<Type>` is shown in Figure 2.5¹.

2.4 Equation discretisation

Equation discretisation converts the PDEs into a set of algebraic equations that are commonly expressed in matrix form as:

$$[A][x] = [b] \quad (2.12)$$

where $[A]$ is a square matrix, $[x]$ is the column vector of dependent variable and $[b]$ is the source vector. The description of $[x]$ and $[b]$ as ‘vectors’ comes from matrix terminology rather than being a precise description of what they truly are: a list of values defined at locations in the geometry, *i.e.* a `geometricField<Type>`, or more specifically a `volField<Type>` when using FV discretisation.

$[A]$ is a list of coefficients of a set of algebraic equations, and cannot be described as a `geometricField<Type>`. It is therefore given a class of its own: `fvMatrix`. `fvMatrix<Type>` is created through discretisation of a `geometric<Type>Field` and therefore inherits the `<Type>`. It supports many of the standard algebraic matrix operations of addition $+$, subtraction $-$ and multiplication $*$.

Each term in a PDE is represented individually in OpenFOAM code using the classes of static functions `finiteVolumeMethod` and `finiteVolumeCalculus`, abbreviated by a `typedef`

¹The diagram is not an exact description of the class hierarchy, rather a representation of the general structure leading from some primitive classes to `geometric<Type>Field`.

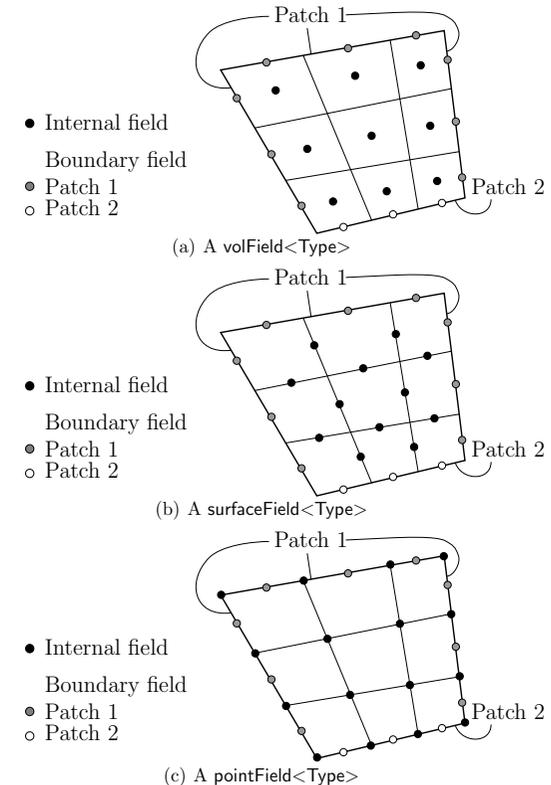
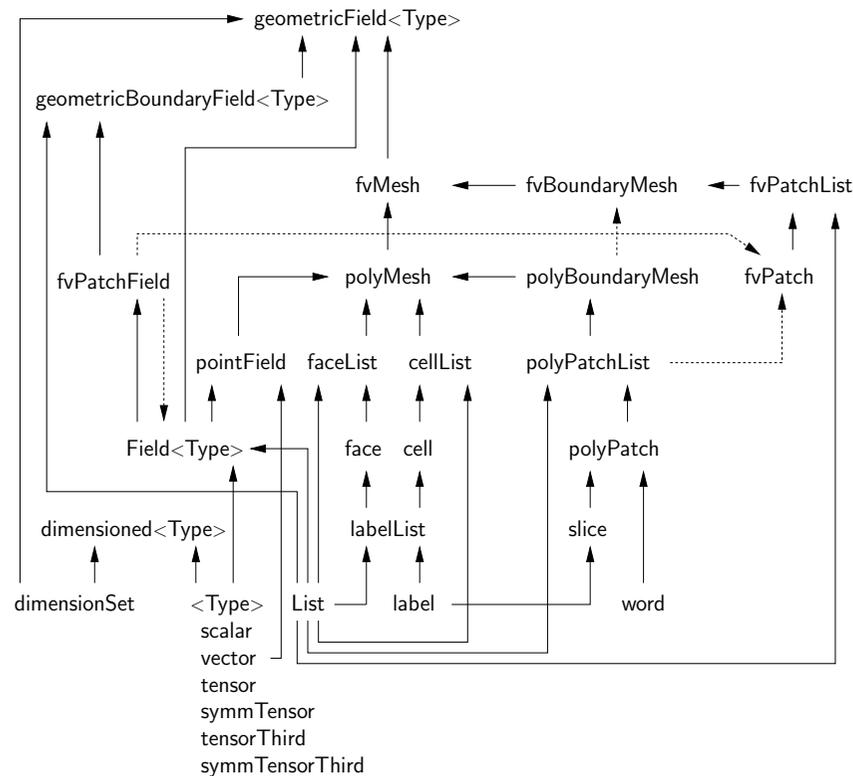


Figure 2.4: Types of `geometricField<Type>` defined on a mesh with 2 boundary patches (in 2 dimensions for simplicity)

Figure 2.5: Basic class structure leading to `geometricField<Type>`

to `fvm` and `fv` respectively. `fvm` and `fv` contain static functions, representing differential operators, *e.g.* ∇^2 , $\nabla \cdot$ and $\partial/\partial t$, that discretise `geometricField<Type>`s. The purpose of defining these functions within two classes, `fvm` and `fv`, rather than one, is to distinguish:

- functions of `fvm` that calculate implicit derivatives of and return an `fvMatrix<Type>`
- some functions of `fv` that calculate explicit derivatives and other explicit calculations, returning a `geometricField<Type>`.

Figure 2.6 shows a `geometricField<Type>` defined on a mesh with 2 boundary patches and illustrates the explicit operations merely transform one field to another and drawn in 2D for simplicity.

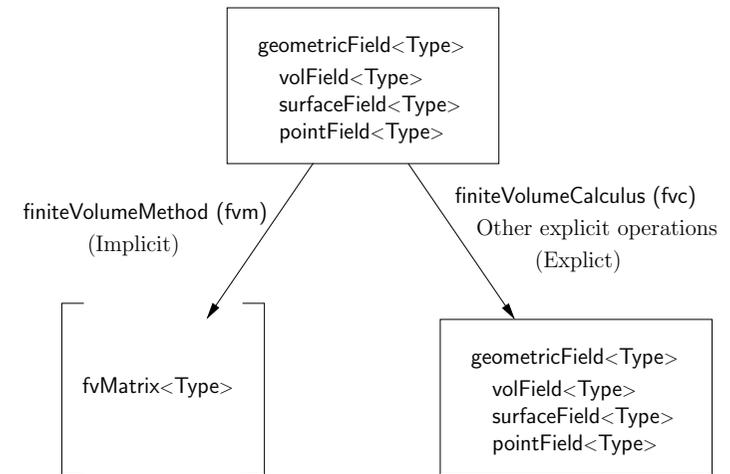
Figure 2.6: A `geometricField<Type>` and its operators

Table 2.2 lists the main functions that are available in `fvm` and `fv` to discretise terms that may be found in a PDE. FV discretisation of each term is formulated by first integrating the term over a cell volume V . Most spatial derivative terms are then converted to integrals over the cell surface S bounding the volume using Gauss's theorem

$$\int_V \nabla \star \phi \, dV = \int_S d\mathbf{S} \star \phi \quad (2.13)$$

where \mathbf{S} is the surface area vector, ϕ can represent any tensor field and the star notation \star is used to represent any tensor product, *i.e.* inner, outer and cross and the respective derivatives: divergence $\nabla \cdot \phi$, gradient $\nabla \phi$ and $\nabla \times \phi$. Volume and surface integrals are then linearised using appropriate schemes which are described for each term in the following Sections. Some terms are always discretised using one scheme, a selection of schemes is offered in OpenFOAM for the discretisation of other terms. The choice of scheme is either made by a direct specification within the code or it can be read from an input file at job run-time and stored within an `fvSchemes` class object.

| Term description | Implicit / Explicit | Text expression | fvm::/fvc:: functions |
|------------------------|------------------------|--|--|
| Laplacian | Imp/Exp | $\nabla^2\phi$ $\nabla \cdot \Gamma \nabla \phi$ | laplacian(phi) laplacian(Gamma, phi) |
| Time derivative | Imp/Exp | $\frac{\partial \phi}{\partial t}$ $\frac{\partial \rho \phi}{\partial t}$ | ddt(phi) ddt(rho, phi) |
| Second time derivative | Imp/Exp | $\frac{\partial}{\partial t} \left(\rho \frac{\partial \phi}{\partial t} \right)$ | d2dt2(rho, phi) |
| Convection | Imp/Exp | $\nabla \cdot (\psi)$ $\nabla \cdot (\psi \phi)$ | div(psi, scheme)* div(psi, phi, word)* div(psi, phi) |
| Divergence | Exp | $\nabla \cdot \chi$ | div(chi) |
| Gradient | Exp | $\nabla \chi$ $\nabla \phi$ | grad(chi) gGrad(phi) lsGrad(phi) snGrad(phi) snGradCorrection(phi) |
| Grad-grad squared | Exp | $ \nabla \nabla \phi ^2$ | sqrGradGrad(phi) |
| Curl | Exp | $\nabla \times \phi$ | curl(phi) |
| Source | Imp Imp/Exp† | $\rho \phi$ | Sp(rho, phi) SuSp(rho, phi) |

†fvm::SuSp source is discretised implicit or explicit depending on the sign of rho.

†An explicit source can be introduced simply as a vol<Type>Field, e.g. rho*phi.

Function arguments can be of the following classes:

phi: vol<Type>Field

Gamma: scalar volScalarField, surfaceScalarField, volTensorField, surfaceTensorField.

rho: scalar, volScalarField

psi: surfaceScalarField.

chi: surface<Type>Field, vol<Type>Field.

Table 2.2: Discretisation of PDE terms in OpenFOAM

2.4.1 The Laplacian term

The Laplacian term is integrated over a control volume and linearised as follows:

$$\int_V \nabla \cdot (\Gamma \nabla \phi) dV = \int_S d\mathbf{S} \cdot (\Gamma \nabla \phi) = \sum_f \Gamma_f \mathbf{S}_f \cdot (\nabla \phi)_f \quad (2.14)$$

The face gradient discretisation is implicit when the length vector \mathbf{d} between the centre of the cell of interest P and the centre of a neighbouring cell N is orthogonal to the face plane, i.e. parallel to \mathbf{S}_f :

$$\mathbf{S}_f \cdot (\nabla \phi)_f = |S_f| \frac{\phi_N - \phi_P}{|\mathbf{d}|} \quad (2.15)$$

In the case of non-orthogonal meshes, an additional explicit term is introduced [?] which is evaluated by interpolating cell centre gradients, themselves calculated by central differencing cell centre values.

2.4.2 The convection term

The convection term is integrated over a control volume and linearised as follows:

$$\int_V \nabla \cdot (\rho \mathbf{U} \phi) dV = \int_S d\mathbf{S} \cdot (\rho \mathbf{U} \phi) = \sum_f \mathbf{S}_f \cdot (\rho \mathbf{U})_f \phi_f = \sum_f F \phi_f \quad (2.16)$$

The face field ϕ_f can be evaluated using a variety of schemes:

Central differencing (CD) is second-order accurate but unbounded

$$\phi_f = f_x \phi_P + (1 - f_x) \phi_N \quad (2.17)$$

where $f_x \equiv \overline{fN}/\overline{PN}$ where \overline{fN} is the distance between f and cell centre N and \overline{PN} is the distance between cell centres P and N .

Upwind differencing (UD) determines ϕ_f from the direction of flow and is bounded at the expense of accuracy

$$\phi_f = \begin{cases} \phi_P & \text{for } F \geq 0 \\ \phi_N & \text{for } F < 0 \end{cases} \quad (2.18)$$

Blended differencing (BD) schemes combine UD and CD in an attempt to preserve boundedness with reasonable accuracy,

$$\phi_f = (1 - \gamma) (\phi_f)_{UD} + \gamma (\phi_f)_{CD} \quad (2.19)$$

OpenFOAM has several implementations of the Gamma differencing scheme to select the blending coefficient γ [?] but it offers other well-known schemes such as van Leer, SUPERBEE, MINMOD etc..

2.4.3 First time derivative

The first time derivative $\partial/\partial t$ is integrated over a control volume as follows:

$$\frac{\partial}{\partial t} \int_V \rho \phi \, dV \quad (2.20)$$

The term is discretised by simple differencing in time using:

new values $\phi^n \equiv \phi(t + \Delta t)$ at the time step we are solving for;

old values $\phi^o \equiv \phi(t)$ that were stored from the previous time step;

old-old values $\phi^{oo} \equiv \phi(t - \Delta t)$ stored from a time step previous to the last.

One of two discretisation schemes can be declared using the `timeScheme` keyword in the appropriate input file, described in detail in [section 4.4](#) of the User Guide.

Euler implicit scheme, `timeScheme EulerImplicit`, that is first order accurate in time:

$$\frac{\partial}{\partial t} \int_V \rho \phi \, dV = \frac{(\rho_P \phi_P V)^n - (\rho_P \phi_P V)^o}{\Delta t} \quad (2.21)$$

Backward differencing scheme, `timeScheme BackwardDifferencing`, that is second order accurate in time by storing the old-old values and therefore with a larger overhead in data storage than `EulerImplicit`:

$$\frac{\partial}{\partial t} \int_V \rho \phi \, dV = \frac{3(\rho_P \phi_P V)^n - 4(\rho_P \phi_P V)^o + (\rho_P \phi_P V)^{oo}}{2\Delta t} \quad (2.22)$$

2.4.4 Second time derivative

The second time derivative is integrated over a control volume and linearised as follows:

$$\frac{\partial}{\partial t} \int_V \rho \frac{\partial \phi}{\partial t} \, dV = \frac{(\rho_P \phi_P V)^n - 2(\rho_P \phi_P V)^o + (\rho_P \phi_P V)^{oo}}{\Delta t^2} \quad (2.23)$$

It is first order accurate in time.

2.4.5 Divergence

The divergence term described in this Section is strictly an explicit term that is distinguished from the convection term of Section [2.4.2](#), *i.e.* in that it is not the divergence of the product of a velocity and dependent variable. The term is integrated over a control volume and linearised as follows:

$$\int_V \nabla \cdot \phi \, dV = \int_S d\mathbf{S} \cdot \phi = \sum_f \mathbf{S}_f \cdot \phi_f \quad (2.24)$$

The `fv::div` function can take as its argument either a `surface<Type>Field`, in which case ϕ_f is specified directly, or a `vol<Type>Field` which is interpolated to the face by central differencing as described in Section [2.4.10](#):

2.4.6 Gradient

The gradient term is an explicit term that can be evaluated in a variety of ways. The scheme can be evaluated either by selecting the particular grad function relevant to the discretisation scheme, *e.g.* `fv::gGrad`, `fv::lsGrad` *etc.*, or by using the `fv::grad` function combined with the appropriate `timeScheme` keyword in an input file

Gauss integration is invoked using the `fv::grad` function with `timeScheme Gauss` or directly using the `fv::gGrad` function. The discretisation is performed using the standard method of applying Gauss's theorem to the volume integral:

$$\int_V \nabla \phi \, dV = \int_S d\mathbf{S} \phi = \sum_f \mathbf{S}_f \phi_f \quad (2.25)$$

As with the `fv::div` function, the Gaussian integration `fv::grad` function can take either a `surfaceField<Type>` or a `volField<Type>` as an argument.

Least squares method is based on the following idea:

1. a value at point P can be extrapolated to neighbouring point N using the gradient at P ;
2. the extrapolated value at N can be compared to the actual value at N , the difference being the error;
3. if we now minimise the sum of the square of weighted errors at all neighbours of P with the respect to the gradient, then the gradient should be a good approximation.

Least squares is invoked using the `fv::grad` function with `timeScheme leastSquares` or directly using the `fv::lsGrad` function. The discretisation is performed as by first calculating the tensor \mathbf{G} at every point P by summing over neighbours N :

$$\mathbf{G} = \sum_N w_N^2 \mathbf{d}\mathbf{d} \quad (2.26)$$

where \mathbf{d} is the vector from P to N and the weighting function $w_N = 1/|\mathbf{d}|$. The gradient is then evaluated as:

$$(\nabla \phi)_P = \sum_N w_N^2 \mathbf{G}^{-1} \cdot \mathbf{d} (\phi_N - \phi_P) \quad (2.27)$$

Surface normal gradient The gradient normal to a surface $\mathbf{n}_f \cdot (\nabla \phi)_f$ can be evaluated at cell faces using the scheme

$$(\nabla \phi)_f = \frac{\phi_N - \phi_P}{|\mathbf{d}|} \quad (2.28)$$

This gradient is called by the function `fv::snGrad` and returns a `surfaceField<Type>`. The scheme is directly analogous to that evaluated for the Laplacian discretisation scheme in Section [2.4.1](#), and in the same manner, a correction can be introduced to improve the accuracy of this face gradient in the case of non-orthogonal meshes. This correction is called using the function `fv::snGradCorrection` [`Check**`].

2.4.7 Grad-grad squared

The grad-grad squared term is evaluated by: taking the gradient of the field; taking the gradient of the resulting gradient field; and then calculating the magnitude squared of the result. The mathematical expression for grad-grad squared of ϕ is $|\nabla(\nabla\phi)|^2$.

2.4.8 Curl

The curl is evaluated from the gradient term described in Section 2.4.6. First, the gradient is discretised and then the curl is evaluated using the relationship from Equation 2.7, repeated here for convenience

$$\nabla \times \phi = 2 * (\text{skew } \nabla \phi)$$

2.4.9 Source terms

Source terms can be specified in 3 ways

Explicit Every explicit term is a `volField<Type>`. Hence, an explicit source term can be incorporated into an equation simply as a field of values. For example if we wished to solve Poisson's equation $\nabla^2\phi = f$, we would define `phi` and `f` as `volScalarField` and then do

```
solve(fvm::laplacian(phi) == f)
```

Implicit An implicit source term is integrated over a control volume and linearised by

$$\int_V \rho\phi \, dV = \rho_P V_P \phi_P \quad (2.29)$$

Implicit/Explicit The implicit source term changes the coefficient of the diagonal of the matrix. Depending on the sign of the coefficient and matrix terms, this will either increase or decrease diagonal dominance of the matrix. Decreasing the diagonal dominance could cause instability during iterative solution of the matrix equation. Therefore OpenFOAM provides a mixed source discretisation procedure that is implicit when the coefficients that are greater than zero, and explicit for the coefficients less than zero. In mathematical terms the matrix coefficient for node P is $V_P \max(\rho_P, 0)$ and the source term is $V_P \phi_P \min(\rho_P, 0)$.

2.4.10 Other explicit discretisation schemes

There are some other discretisation procedures that convert `volField<Type>`s into `surface<Type>Fields` and visa versa.

Surface integral `fvc::surfaceIntegrate` performs a summation of `surface<Type>Field` face values bounding each cell and dividing by the cell volume, *i.e.* $(\sum_f \phi_f)/V_P$. It returns a `volField<Type>`.

Surface sum `fvc::surfaceSum` performs a summation of `surface<Type>Field` face values bounding each cell, *i.e.* $\sum_f \phi_f$ returning a `volField<Type>`.

Average `fvc::average` produces an area weighted average of `surface<Type>Field` face values, *i.e.* $(\sum_f S_f \phi_f)/\sum_f S_f$, and returns a `volField<Type>`.

Reconstruct

Face interpolate The `geometric<Type>Field` function `faceInterpolate()` interpolates `volField<Type>` cell centre values to cell faces using central differencing, returning a `surface<Type>Field`.

2.5 Temporal discretisation

Although we have described the discretisation of temporal derivatives in Sections 2.4.3 and 2.4.4, we need to consider how to treat the spatial derivatives in a transient problem. If we denote all the spatial terms as $\mathcal{A}\phi$ where \mathcal{A} is any spatial operator, *e.g.* Laplacian, then we can express a transient PDE in integral form as

$$\int_t^{t+\Delta t} \left[\frac{\partial}{\partial t} \int_V \rho\phi \, dV + \int_V \mathcal{A}\phi \, dV \right] dt = 0 \quad (2.30)$$

Using the Euler implicit method of Equation 2.21, the first term can be expressed as

$$\begin{aligned} \int_t^{t+\Delta t} \left[\frac{\partial}{\partial t} \int_V \rho\phi \, dV \right] dt &= \int_t^{t+\Delta t} \frac{(\rho_P \phi_P V)^n - (\rho_P \phi_P V)^o}{\Delta t} dt \\ &= \frac{(\rho_P \phi_P V)^n - (\rho_P \phi_P V)^o}{\Delta t} \Delta t \end{aligned} \quad (2.31)$$

The second term can be expressed as

$$\int_t^{t+\Delta t} \left[\int_V \mathcal{A}\phi \, dV \right] dt = \int_t^{t+\Delta t} \mathcal{A}^* \phi \, dt \quad (2.32)$$

where \mathcal{A}^* represents the spatial discretisation of \mathcal{A} . The time integral can be discretised in three ways:

Euler implicit uses implicit discretisation of the spatial terms, thereby taking current values ϕ^n .

$$\int_t^{t+\Delta t} \mathcal{A}^* \phi \, dt = \mathcal{A}^* \phi^n \Delta t \quad (2.33)$$

It is first order accurate in time, guarantees boundedness and is unconditionally stable.

Explicit uses explicit discretisation of the spatial terms, thereby taking old values ϕ^o .

$$\int_t^{t+\Delta t} \mathcal{A}^* \phi \, dt = \mathcal{A}^* \phi^o \Delta t \quad (2.34)$$

It is first order accurate in time and is unstable if the Courant number Co is greater than 1. The Courant number is defined as

$$Co = \frac{\mathbf{U}_f \cdot \mathbf{d}}{|\mathbf{d}|^2 \Delta t} \quad (2.35)$$

where \mathbf{U}_f is a characteristic velocity, *e.g.* velocity of a wave front, velocity of flow.

Crank Nicholson uses the trapezoid rule to discretise the spatial terms, thereby taking a mean of current values ϕ^n and old values ϕ^o .

$$\int_t^{t+\Delta t} \mathcal{A}^* \phi \, dt = \mathcal{A}^* \left(\frac{\phi^n + \phi^o}{2} \right) \Delta t \quad (2.36)$$

It is second order accurate in time, is unconditionally stable but does not guarantee boundedness.

2.5.1 Treatment of temporal discretisation in OpenFOAM

At present the treatment of the temporal discretisation is controlled by the implementation of the spatial derivatives in the PDE we wish to solve. For example, let us say we wish to solve a transient diffusion equation

$$\frac{\partial \phi}{\partial t} = \kappa \nabla^2 \phi \quad (2.37)$$

An Euler implicit implementation of this would read

```
solve(fvm::ddt(phi) == kappa*fvm::laplacian(phi))
```

where we use the `fvm` class to discretise the `Laplacian` term implicitly. An explicit implementation would read

```
solve(fvm::ddt(phi) == kappa*fvc::laplacian(phi))
```

where we now use the `fvc` class to discretise the `Laplacian` term explicitly. The Crank Nicholson scheme can be implemented by the mean of implicit and explicit terms:

```
solve
(
  fvm::ddt(phi)
  ==
  kappa*0.5*(fvm::laplacian(phi) + fvc::laplacian(phi))
)
```

2.6 Boundary Conditions

Boundary conditions are required to complete the problem we wish to solve. We therefore need to specify boundary conditions on all our boundary faces. Boundary conditions can be divided into 2 types:

Dirichlet prescribes the value of the dependent variable on the boundary and is therefore termed ‘fixed value’ in this guide;

Neumann prescribes the gradient of the variable normal to the boundary and is therefore termed ‘fixed gradient’ in this guide.

When we perform discretisation of terms that include the sum over faces \sum_f , we need to consider what happens when one of the faces is a boundary face.

Fixed value We specify a fixed value at the boundary ϕ_b

- We can simply substitute ϕ_b in cases where the discretisation requires the value on a boundary face ϕ_f , *e.g.* in the convection term in [Equation 2.16](#).
- In terms where the face gradient $(\nabla\phi)_f$ is required, *e.g.* Laplacian, it is calculated using the boundary face value and cell centre value,

$$\mathbf{S}_f \cdot (\nabla\phi)_f = |S_f| \frac{\phi_b - \phi_P}{|\mathbf{d}|} \quad (2.38)$$

Fixed gradient The fixed gradient boundary condition g_b is a specification on inner product of the gradient and unit normal to the boundary, or

$$g_b = \left(\frac{\mathbf{S}}{|\mathbf{S}|} \cdot \nabla\phi \right)_f \quad (2.39)$$

- When discretisation requires the value on a boundary face ϕ_f we must interpolate the cell centre value to the boundary by

$$\begin{aligned} \phi_f &= \phi_P + \mathbf{d} \cdot (\nabla\phi)_f \\ &= \phi_P + |\mathbf{d}| g_b \end{aligned} \quad (2.40)$$

- ϕ_b can be directly substituted in cases where the discretisation requires the face gradient to be evaluated,

$$\mathbf{S}_f \cdot (\nabla\phi)_f = |S_f| g_b \quad (2.41)$$

2.6.1 Physical boundary conditions

The specification of boundary conditions is usually an engineer’s interpretation of the true behaviour. Real boundary conditions are generally defined by some physical attributes rather than the numerical description as described of the previous Section. In incompressible fluid flow there are the following physical boundaries

Inlet The velocity field at the inlet is supplied and, for consistency, the boundary condition on pressure is zero gradient.

Outlet The pressure field at the outlet is supplied and a zero gradient boundary condition on velocity is specified.

No-slip impermeable wall The velocity of the fluid is equal to that of the wall itself, *i.e.* a fixed value condition can be specified. The pressure is specified zero gradient since the flux through the wall is zero.

In a problem whose solution domain and boundary conditions are symmetric about a plane, we only need to model half the domain to one side of the symmetry plane. The boundary condition on the plane must be specified according to

Symmetry plane The symmetry plane condition specifies the component of the gradient normal to the plane should be zero. [Check**]

Chapter 3

Examples of the use of OpenFOAM

In this section we shall describe several test cases supplied with the OpenFOAM distribution. The intention is to provide example cases, including those in the tutorials in [chapter 2](#) of the User Guide, for every standard solver. The examples are designed to introduce certain tools and features of OpenFOAM, *e.g.* within pre-/post-processing, numerical schemes, algorithms. They also provide a means for validation of solvers although that is not their principal function.

Each example contains a description of the problem: the geometry, initial and boundary conditions, a brief description of the equations being solved, models used, and physical properties required. The solution domain is selected which may be a portion of the original geometry, *e.g.* if we introduce symmetry planes. The method of meshing, usually `blockMesh`, is specified; of course the user can simply view the mesh since every example is distributed with the `polyMesh` directory containing the data files that describe the mesh.

The examples coexist with the tutorials in the `tutorials` subdirectory of the OpenFOAM installation. They are organised into a set of subdirectories by solver, *e.g.* all the `icoFoam` cases are stored within a subdirectory `icoFoam`. Before running a particular example, the user is urged to copy it into their user account. We recommend that the user stores all OpenFOAM cases in a directory we recommend that the tutorials are copied into a directory `$FOAM_RUN`. If this directory structure has not yet been created in the user's account, it can be created with

```
mkdir -p $FOAM_RUN
```

The tutorials can then be copied into this directory with

```
cp -r $FOAM_TUTORIALS/* $FOAM_RUN
```

3.1 Flow around a cylinder

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

- non-orthogonal meshes;
- generating an analytical solution to a problem in OpenFOAM.

3.1.1 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 3.1](#).

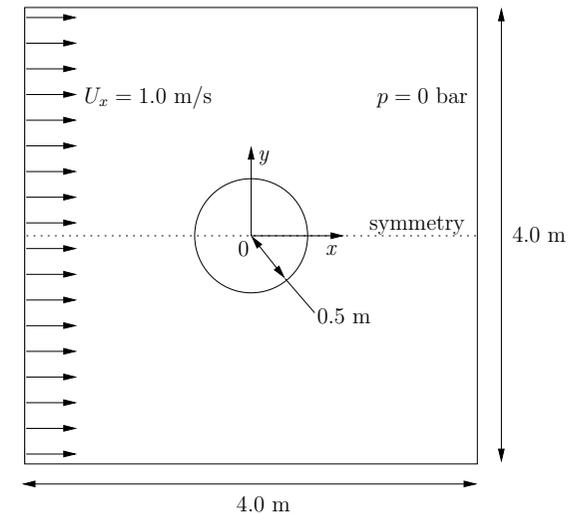


Figure 3.1: Geometry of flow round a cylinder

Governing equations

- Mass continuity for an incompressible fluid

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

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

$$\nabla^2 p = 0 \quad (3.2)$$

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/potentialFoam` directory.

3.1.2 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.

3.1.3 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 3.2](#). Remember that all

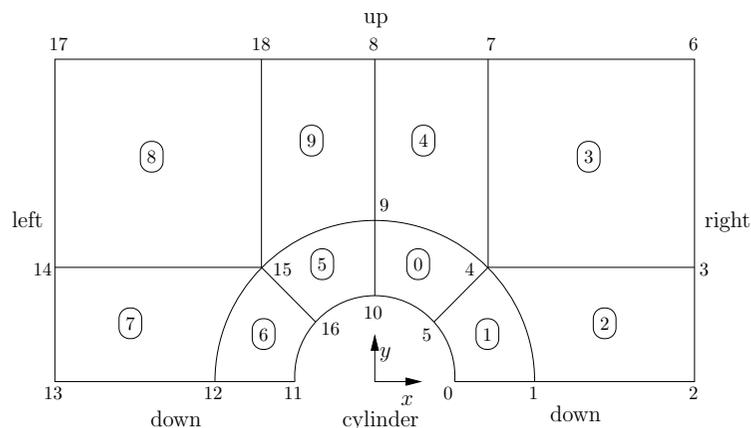


Figure 3.2: Blocks in cylinder geometry

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 3.2](#) 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  /*-----*
2  |  F i e l d   |   O p e n F O A M :   T h e   O p e n   S o u r c e   C F D   T o o l b o x
3  |  O p e r a t i o n   |   V e r s i o n :   1 . 3
4  |  A n d   |   W e b :   h t t p : / / w w w . o p e n f o a m . o r g
5  |  M a n i p u l a t i o n   |
6  |-----*
7  */
8
9  FoamFile
10 {
11     version      2.0;
12     format       ascii;
13
14     root         "";
15     case        "";
16     instance    "";
17     local      "";
18 }

```

```

19     class        dictionary;
20     object       blockMeshDict;
21 }
22
23 // *****
24
25 convertToMeters 1;
26
27 vertices
28 (
29     (0.5 0 -0.5)
30     (1 0 -0.5)
31     (2 0 -0.5)
32     (2 0.707107 -0.5)
33     (0.707107 0.707107 -0.5)
34     (0.353553 0.353553 -0.5)
35     (2 -0.5)
36     (0.707107 2 -0.5)
37     (0 2 -0.5)
38     (0 1 -0.5)
39     (0 0.5 -0.5)
40     (-0.5 0 -0.5)
41     (-1 0 -0.5)
42     (-2 0 -0.5)
43     (-2 0.707107 -0.5)
44     (-0.707107 0.707107 -0.5)
45     (-0.353553 0.353553 -0.5)
46     (-2 2 -0.5)
47     (-0.707107 2 -0.5)
48     (0.5 0 0.5)
49     (1 0 0.5)
50     (2 0 0.5)
51     (2 0.707107 0.5)
52     (0.707107 0.707107 0.5)
53     (0.353553 0.353553 0.5)
54     (2 2 0.5)
55     (0.707107 2 0.5)
56     (0 2 0.5)
57     (0 1 0.5)
58     (0 0.5 0.5)
59     (-0.5 0 0.5)
60     (-1 0 0.5)
61     (-2 0 0.5)
62     (-2 0.707107 0.5)
63     (-0.707107 0.707107 0.5)
64     (-0.353553 0.353553 0.5)
65     (-2 2 0.5)
66     (-0.707107 2 0.5)
67 );
68
69 blocks
70 (
71     hex (5 4 9 10 24 23 28 29) (10 10 1) simpleGrading (1 1 1)
72     hex (0 1 4 5 19 20 23 24) (10 10 1) simpleGrading (1 1 1)
73     hex (1 2 3 4 20 21 22 23) (20 10 1) simpleGrading (1 1 1)
74     hex (4 3 6 7 23 22 25 26) (20 20 1) simpleGrading (1 1 1)
75     hex (9 4 7 8 28 23 26 27) (10 20 1) simpleGrading (1 1 1)
76     hex (15 16 10 9 34 35 29 28) (10 10 1) simpleGrading (1 1 1)
77     hex (12 11 16 15 31 30 35 34) (10 10 1) simpleGrading (1 1 1)
78     hex (13 12 15 14 32 31 34 33) (20 10 1) simpleGrading (1 1 1)
79     hex (14 15 18 17 33 34 37 36) (20 20 1) simpleGrading (1 1 1)
80     hex (15 9 8 18 34 28 27 37) (10 20 1) simpleGrading (1 1 1)
81 );
82
83 edges
84 (
85     arc 0 5 (0.469846 0.17101 -0.5)
86     arc 5 10 (0.17101 0.469846 -0.5)
87     arc 1 4 (0.939693 0.34202 -0.5)
88     arc 4 9 (0.34202 0.939693 -0.5)
89     arc 19 24 (0.469846 0.17101 0.5)
90     arc 24 29 (0.17101 0.469846 0.5)
91     arc 20 23 (0.939693 0.34202 0.5)
92     arc 23 28 (0.34202 0.939693 0.5)
93     arc 11 16 (-0.469846 0.17101 -0.5)
94     arc 16 10 (-0.17101 0.469846 -0.5)
95     arc 12 15 (-0.939693 0.34202 -0.5)
96     arc 15 9 (-0.34202 0.939693 -0.5)
97     arc 30 35 (-0.469846 0.17101 0.5)
98     arc 35 29 (-0.17101 0.469846 0.5)
99     arc 31 34 (-0.939693 0.34202 0.5)
100    arc 34 28 (-0.34202 0.939693 0.5)
101 );
102

```

```

103 patches
104 (
105     symmetryPlane down
106     (
107         (0 1 20 19)
108         (1 2 21 20)
109         (12 11 30 31)
110         (13 12 31 32)
111     )
112     patch right
113     (
114         (2 3 22 21)
115         (3 6 25 22)
116     )
117     symmetryPlane up
118     (
119         (7 8 27 26)
120         (6 7 26 25)
121         (8 18 37 27)
122         (18 17 36 37)
123     )
124     patch left
125     (
126         (14 13 32 33)
127         (17 14 33 36)
128     )
129     symmetryPlane cylinder
130     (
131         (10 5 24 29)
132         (5 0 19 24)
133         (16 10 29 35)
134         (11 16 35 30)
135     )
136 );
137
138 mergePatchPairs
139 (
140 );
141 // ***** //
142

```

3.1.4 Boundary conditions and initial fields

Using FoamX or editing case files by hand, set the boundary conditions in accordance with the problem description in [Figure 3.1](#), *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.

3.1.5 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  |-----*-----|
2  | |=====| F ield | OpenFOAM: The Open Source CFD Toolbox
3  | | \ / \ / | O peration | Version: 1.3
4  | |  \ /  \ | A nd | Web: http://www.openfoam.org
5  | |  \ /  \ | M anipulation |
6  | |-----*-----|
7
8
9  FoamFile
10 {
11     version      2.0;

```

```

12     format      ascii;
13
14     root        "";
15     case        "";
16     instance    "";
17     local       "";
18
19     class        dictionary;
20     object       controlDict;
21 }
22
23 // ***** //
24
25 application potentialFoam;
26
27 startFrom      startTime;
28
29 startTime      0;
30
31 stopAt         endTime;
32
33 endTime        1;
34
35 deltaT         1;
36
37 writeControl   timeStep;
38
39 writeInterval  1;
40
41 purgeWrite     0;
42
43 writeFormat    ascii;
44
45 writePrecision 6;
46
47 writeCompression uncompressed;
48
49 timeFormat     general;
50
51 timePrecision  6;
52
53 runtimeModifiable yes;
54
55 // ***** //
56

```

`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 `controlDict`. 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 3.3\(a\)](#) (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 3.3\(c\)](#), 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 3.3\(b\)](#).

3.1.6 Generating the analytical solution

Source code is included in the `$FOAM_TUTORIALS/potentialFoam/analyticalCylinder` directory to generate the analytical solution for the potential flow case. The velocity at any point at a distance d and angle θ from the cylinder centre is described analytically as

$$U_x = U_\infty \left[1 - \left(\frac{r}{d} \right)^2 \cos 2\theta \right]$$

$$U_y = U_\infty \left(\frac{r}{d} \right)^2 \sin 2\theta \quad (3.3)$$

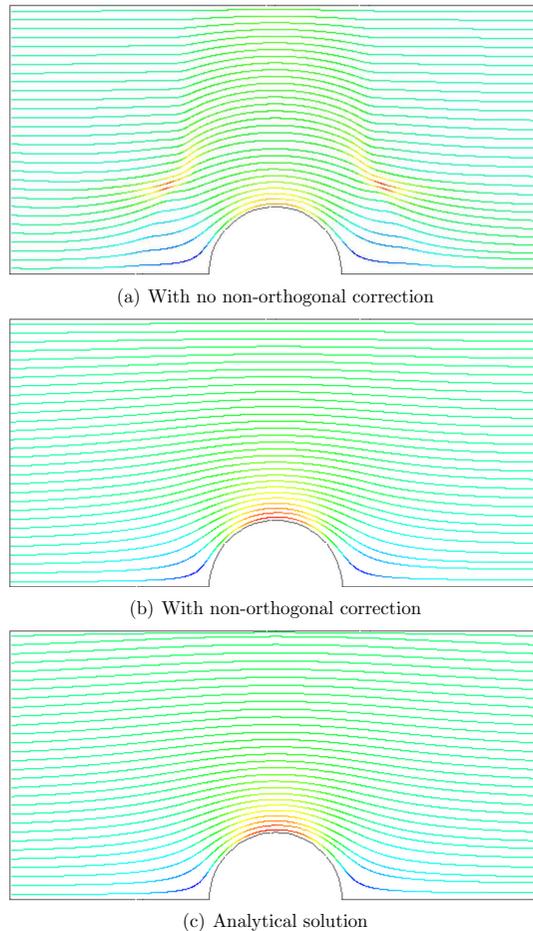


Figure 3.3: Streamlines of potential flow

where r is the cylinder radius and U_∞ is the inlet flow velocity. Here, θ describes the angle subtended from the x -axis.

Let us examine some details of the source code in the *analyticalCylinder* directory. In *createFields.H*, the velocity field is read in using the `IObject::NO_WRITE` option to ensure that the field data can never be overwritten during execution of *analyticalCylinder*. The inlet velocity and cylinder radius are taken from data read from the mesh and a field `UA` is set up to store the analytical solution:

```

1  Info<< "Reading field U\n" << endl;
2  volVectorField U
3  (
4      IObject
5      (
6          "U",
7          runtime.timeName(),
8          mesh,
9          IObject::MUST_READ,
10         IObject::NO_WRITE
11     ),
12     mesh
13 );
14
15 Info<< "Reading inlet velocity uInfx\n" << endl;
16
17 dimensionedScalar uInfx
18 (
19     "uInfx",
20     dimensionSet(0, 1, -1, 0, 0),
21     U.boundaryField()[3][0].x()
22 );
23 Info << "U at inlet = " << uInfx.value() << " m/s" << endl;
24
25 dimensionedScalar radius
26 (
27     "radius",
28     dimensionSet(0, 1, 0, 0, 0),
29     mag(U.mesh().boundary()[4].Cf()[0])
30 );
31
32 Info << "Cylinder radius = " << radius.value() << " m" << endl;
33
34 volVectorField UA
35 (
36     IObject
37     (
38         "UA",
39         runtime.timeName(),
40         mesh,
41         IObject::NO_READ,
42         IObject::AUTO_WRITE
43     ),
44     U
45 );

```

The main code *analyticalCylinder.C* performs the following tasks:

- increments the time step by `runTime++`;
- generates the analytical solution for field `UA` using tensor arithmetic;
- writes the solution to file by `runTime.writeObjects()`.

```

1  /*-----*
2  ===== F i e l d           |   OpenFOAM: The Open Source CFD Toolbox
3  \ \ \ \ \ O p e r a t i o n   |   |
4  \ \ \ \ \ A n d              |   Copyright (C) 1991-2005 OpenCFD Ltd.
5  \ \ \ \ \ M a n i p u l a t i o n |   |
6  -----*
7
8  License
9  This file is part of OpenFOAM.
10
11  OpenFOAM is free software; you can redistribute it and/or modify it
12  under the terms of the GNU General Public License as published by the
13  Free Software Foundation; either version 2 of the License, or (at your
14  option) any later version.

```

```

15
16 OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
17 ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
18 FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
19 for more details.
20
21 You should have received a copy of the GNU General Public License
22 along with OpenFOAM; if not, write to the Free Software Foundation,
23 Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
24
25 Application
26 analyticalCylinder
27
28 Description
29 Generates an analytical solution for potential flow around a cylinder.
30 Can be compared with the solution from the potentialFlow/cylinder example.
31
32 \*-----*/
33
34 #include "fvCFD.H"
35
36 // * * * * *
37
38 int main(int argc, char *argv[])
39 {
40
41 # include "setRootCase.H"
42
43 # include "createTime.H"
44 # include "createMesh.H"
45 # include "createFields.H"
46
47 // * * * * *
48
49 Info << "\nEvaluating analytical solution" << endl;
50
51 volVectorField centres = UA.mesh().C();
52 volScalarField magCentres = mag(centres);
53 volScalarField theta = acos((centres & vector(1,0,0))/magCentres);
54
55 volVectorField cs2theta =
56     cos(2*theta)*vector(1,0,0)
57     + sin(2*theta)*vector(0,1,0);
58
59 UA = uInfX*(dimensionedVector(vector(1,0,0))
60     - pow((radius/magCentres),2)*cs2theta);
61
62 runtime.write();
63
64 Info<< "end" << endl;
65
66 return(0);
67 }
68
69 // * * * * *
70

```

The utility must be compiled with `wmake` as normal. It can then be run by typing

```
analyticalCylinder $FOAM_RUN/potentialFoam cylinder
```

The analytical solution is plotted as streamlines as shown in [Figure 3.3\(c\)](#). Note that differences in the analytical and numerical solutions at the top plane are due to the fact that the analytical solution assumes an infinite boundary and the numerical solution specifies a `zeroGradient` boundary condition at that boundary.

3.1.7 Exercise

Investigate the accuracy of the numerical solution by implementing some measure of comparison between the numerical and analytical in `analyticalCylinder`.

3.2 Steady turbulent flow over a backward-facing step

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.

3.2.1 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 3.4](#).

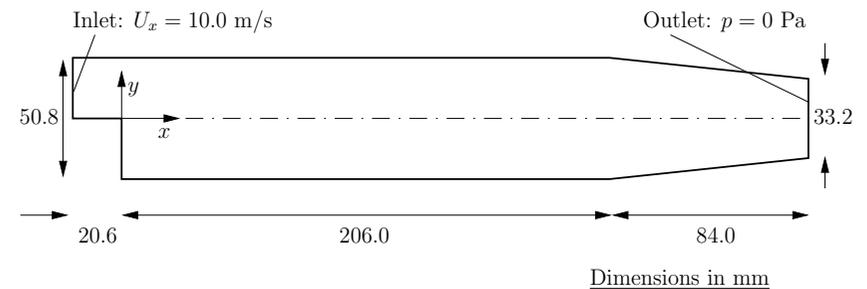


Figure 3.4: Geometry of backward-facing step

Governing equations

- Mass continuity for incompressible flow

$$\nabla \cdot \mathbf{U} = 0 \quad (3.4)$$

- Steady flow momentum equation

$$\nabla \cdot (\mathbf{UU}) + \nabla \cdot \mathbf{R} = -\nabla p \quad (3.5)$$

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

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/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 in incompressibleTransportModels library of the standard OpenFOAM release.

3.2.2 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 3.5.

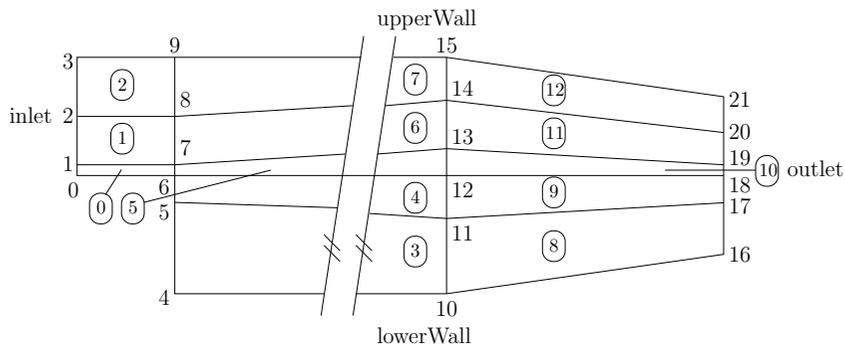


Figure 3.5: Blocks in backward-facing step

The mesh is 3 dimensional, as always in OpenFOAM, so in Figure 3.5 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 |-----*-----|
2 |=====|
3 | \ \ / F ield | OpenFOAM: The Open Source CFD Toolbox
4 | \ \ / O peration | Version: 1.3

```

```

5 | \ \ / A nd | Web: http://www.openfoam.org
6 | \ \ / M anipulation |
7 |-----*-----|
8
9 FoamFile
10 {
11     version      2.0;
12     format       ascii;
13
14     root         "";
15     case        "";
16     instance    "";
17     local       "";
18
19     class        dictionary;
20     object       blockMeshDict;
21 }
22
23 // *****
24
25 convertToMeters 0.001;
26
27 vertices
28 (
29     (-20.6 0 -0.5)
30     (-20.6 3 -0.5)
31     (-20.6 12.7 -0.5)
32     (-20.6 25.4 -0.5)
33     (0 -25.4 -0.5)
34     (0 -5 -0.5)
35     (0 0 -0.5)
36     (0 3 -0.5)
37     (0 12.7 -0.5)
38     (0 25.4 -0.5)
39     (206 -25.4 -0.5)
40     (206 -8.5 -0.5)
41     (206 0 -0.5)
42     (206 6.5 -0.5)
43     (206 17 -0.5)
44     (206 25.4 -0.5)
45     (290 -16.6 -0.5)
46     (290 -6.3 -0.5)
47     (290 0 -0.5)
48     (290 4.5 -0.5)
49     (290 11 -0.5)
50     (290 16.6 -0.5)
51     (-20.6 0 0.5)
52     (-20.6 3 0.5)
53     (-20.6 12.7 0.5)
54     (-20.6 25.4 0.5)
55     (0 -25.4 0.5)
56     (0 -5 0.5)
57     (0 0 0.5)
58     (0 3 0.5)
59     (0 12.7 0.5)
60     (0 25.4 0.5)
61     (206 -25.4 0.5)
62     (206 -8.5 0.5)
63     (206 0 0.5)
64     (206 6.5 0.5)
65     (206 17 0.5)
66     (206 25.4 0.5)
67     (290 -16.6 0.5)
68     (290 -6.3 0.5)
69     (290 0 0.5)
70     (290 4.5 0.5)
71     (290 11 0.5)
72     (290 16.6 0.5)
73 );
74
75 blocks
76 (
77     hex (0 6 7 1 22 28 29 23) (18 7 1) simpleGrading (0.5 1.8 1)
78     hex (1 7 8 2 23 29 30 24) (18 10 1) simpleGrading (0.5 4 1)
79     hex (2 8 9 3 24 30 31 25) (18 13 1) simpleGrading (0.5 0.25 1)
80     hex (4 10 11 5 26 32 33 27) (180 18 1) simpleGrading (4 1 1)
81     hex (5 11 12 6 27 33 34 28) (180 9 1) edgeGrading (4 4 4 4 0.5 1 1 0.5 1 1 1 1)
82     hex (6 12 13 7 28 34 35 29) (180 7 1) edgeGrading (4 4 4 4 1.8 1 1 1.8 1 1 1 1)
83     hex (7 13 14 8 29 35 36 30) (180 10 1) edgeGrading (4 4 4 4 1 1 4 1 1 1 1 1)
84     hex (8 14 15 9 30 36 37 31) (180 13 1) simpleGrading (4 0.25 1)
85     hex (10 16 17 11 32 38 39 33) (25 18 1) simpleGrading (2.5 1 1)
86     hex (11 17 18 12 33 39 40 34) (25 9 1) simpleGrading (2.5 1 1)
87     hex (12 18 19 13 34 40 41 35) (25 7 1) simpleGrading (2.5 1 1)
88     hex (13 19 20 14 35 41 42 36) (25 10 1) simpleGrading (2.5 1 1)

```

```

89     hex (14 20 21 15 36 42 43 37) (25 13 1) simpleGrading (2.5 0.25 1)
90 );
91
92 edges
93 (
94 );
95
96 patches
97 (
98     patch inlet
99     (
100         (0 22 23 1)
101         (1 23 24 2)
102         (2 24 25 3)
103     )
104     patch outlet
105     (
106         (16 17 39 38)
107         (17 18 40 39)
108         (18 19 41 40)
109         (19 20 42 41)
110         (20 21 43 42)
111     )
112     wall upperWall
113     (
114         (3 25 31 9)
115         (9 31 37 15)
116         (15 37 43 21)
117     )
118     wall lowerWall
119     (
120         (0 6 28 22)
121         (6 5 27 28)
122         (5 4 26 27)
123         (4 10 32 26)
124         (10 16 38 32)
125     )
126     empty frontAndBack
127     (
128         (22 28 29 23)
129         (23 29 30 24)
130         (24 30 31 25)
131         (26 32 33 27)
132         (27 33 34 28)
133         (28 34 35 29)
134         (29 35 36 30)
135         (30 36 37 31)
136         (32 38 39 33)
137         (33 39 40 34)
138         (34 40 41 35)
139         (35 41 42 36)
140         (36 42 43 37)
141         (0 1 7 6)
142         (1 2 8 7)
143         (2 3 9 8)
144         (4 5 11 10)
145         (5 6 12 11)
146         (6 7 13 12)
147         (7 8 14 13)
148         (8 9 15 14)
149         (10 11 17 16)
150         (11 12 18 17)
151         (12 13 19 18)
152         (13 14 20 19)
153         (14 15 21 20)
154     )
155 );
156
157 mergePatchPairs
158 (
159 );
160
161 // ***** //

```

A major feature of this problem is the use of the full mesh grading capability of `blockMesh` that is described in [section 6.3.1](#) 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 6.3.1](#) 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 3.5](#). 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 or from within `FoamX` and viewed as described in previous examples.

3.2.3 Boundary conditions and initial fields

The case files can be viewed, or edited from within `FoamX` or by hand. In this case, we are required to set the initial and boundary fields for velocity \mathbf{U} , pressure p , turbulent kinetic energy k and dissipation rate ϵ . The boundary conditions can be specified by setting the physical patch types in `FoamX`: 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 ϵ . \mathbf{U} is given in the problem specification, but the values of k and ϵ 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

$$U'_x = U'_y = U'_z = \frac{5}{100} 10 = 0.5 \text{ m/s} \quad (3.6)$$

and

$$k = \frac{3}{2} (0.5)^2 = 0.375 \text{ m}^2/\text{s}^2 \quad (3.7)$$

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

$$\epsilon = \frac{C_\mu^{0.75} k^{1.5}}{l} = \frac{0.09^{0.75} 0.375^{1.5}}{0.1 \times 25.4 \times 10^{-3}} = 14.855 \text{ m}^2/\text{s}^3 \quad (3.8)$$

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

3.2.4 Case control

The choices of `fvSchemes` are as follows: the `timeScheme` should be `SteadyState`; the `gradScheme` and `laplacianScheme` should be set as default to `Gauss`; and, the `divScheme` should be set to `UD` to ensure boundedness.

Special attention should be paid to the settings of `fvTolerances`. Although the top level `simpleFoam` code contains only equations for p and \mathbf{U} , the turbulent model solves equations for k , ϵ and \mathbf{R} , and tolerance settings are required for all 5 equations. A `solverTolerance` of 10^{-5} and `solverRelativeTolerance` 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 , ϵ and \mathbf{R} but 0.3 is required for p to avoid numerical instability.

Finally, in `controlDict`, 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 `writeFrequency` is sufficiently high, *e.g.* 50, that you will not fill the hard disk with data during run time.

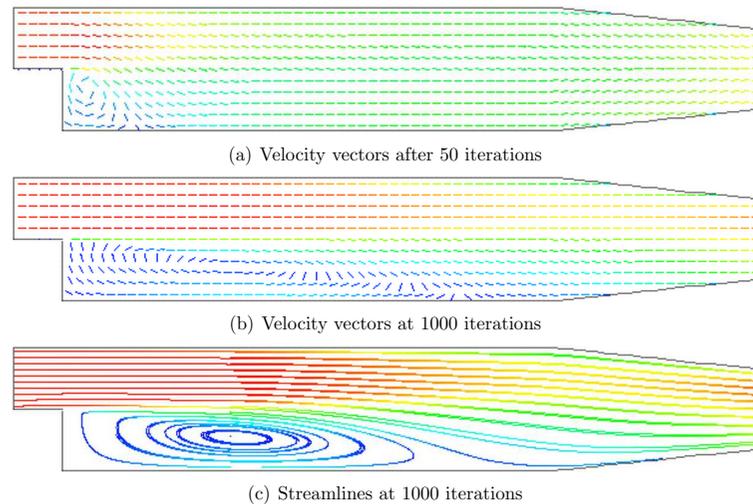


Figure 3.6: Development of a vortex in the backward-facing step.

3.2.5 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 3.6\(a\)](#). Over several iterations 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 3.6\(b-c\)](#).

3.3 Supersonic flow over a forward-facing step

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;

3.3.1 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 3.7](#)

Governing equations

- Mass continuity

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \quad (3.9)$$

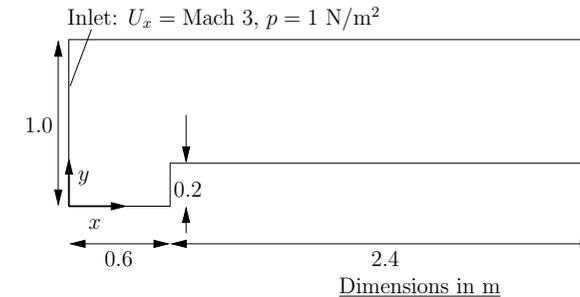


Figure 3.7: Geometry of the forward step geometry

- Ideal gas

$$p = \rho RT \quad (3.10)$$

- Momentum equation for Newtonian fluid

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

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

$$\frac{\partial \rho e}{\partial t} + \nabla \cdot (\rho \mathbf{U} e) - \nabla \cdot \left(\frac{k}{C_v} \right) \nabla e = p \nabla \cdot \mathbf{U} \quad (3.12)$$

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/sonicFoam` directory.

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

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

- Mesh refinement
- Pressure waves in liquids

3.4.1 Problem specification

Solution domain The domain is 2 dimensional and consists of a tank with a small outflow pipe as shown in [Figure 3.9](#)

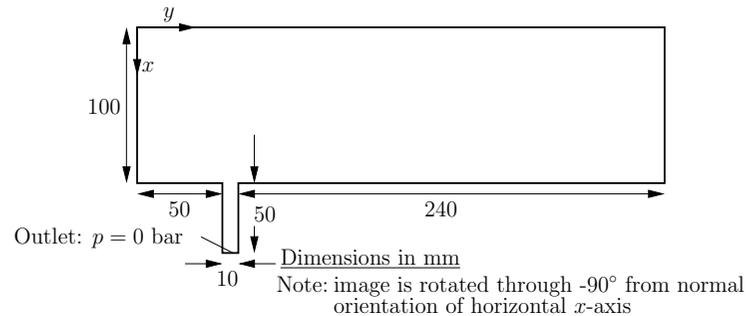


Figure 3.9: Geometry of a tank with outflow pipe

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

- Mass continuity

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \quad (3.14)$$

- The barotropic relationship

$$\frac{\partial \rho}{\partial p} = \frac{\rho}{K} = \psi \quad (3.15)$$

where K is the bulk modulus

- [Equation 3.15](#) is linearised as

$$\rho \approx \rho_0 + \psi (p - p_0) \quad (3.16)$$

where ρ_0 and p_0 are the reference density and pressure respectively such that $\rho(p_0) = \rho_0$.

- Momentum equation for Newtonian fluid

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot (\rho \mathbf{U} \mathbf{U}) - \nabla \cdot \mu \nabla \mathbf{U} = -\nabla p \quad (3.17)$$

Boundary conditions Using FoamX the following physical boundary conditions can be set:

- `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 $\mathbf{U} = 0$ m/s, $p = 100$ bar.

Transport properties

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

Thermodynamic properties

- Density of water $\rho = 1000$ kg/m³
- Reference pressure $p_0 = 1$ bar
- Compressibility of water $\psi = 4.54 \times 10^{-7}$ s²/m²

Solver name `sonicLiquidFoam`: a compressible sonic laminar liquid flow code.

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

3.4.2 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  /*-----*
2  |=====| F ield      | OpenFOAM: The Open Source CFD Toolbox
3  | \ \ \ \ | O peration | Version: 1.3
4  |  \ \ \  | A nd       | Web: http://www.openfoam.org
5  |   \ \   | M anipulation |
6  |-----*
7
8
9  FoamFile
10 {
11     version      2.0;
12     format       ascii;
13
14     root         "";
15     case        "";
16     instance    "";
17     local       "";
18
19     class        dictionary;
20     object       blockMeshDict;
21 }
22
23 // *****
24
25 convertToMeters 0.1;
26
27 vertices
28 (
29     (0 0 -0.1)
30     (1 0 -0.1)
31     (0 0.5 -0.1)
32     (1 0.5 -0.1)
33     (1.5 0.5 -0.1)
34     (0 0.6 -0.1)
35     (1 0.6 -0.1)
36     (1.5 0.6 -0.1)
37     (0 3 -0.1)
38     (1 3 -0.1)
39     (0 0 0.1)
40     (1 0 0.1)
41     (0 0.5 0.1)

```

```

42     (1 0.5 0.1)
43     (1.5 0.5 0.1)
44     (0 0.6 0.1)
45     (1 0.6 0.1)
46     (1.5 0.6 0.1)
47     (0 3 0.1)
48     (1 3 0.1)
49 );
50
51 blocks
52 (
53     hex (0 1 3 2 10 11 13 12) (30 20 1) simpleGrading (1 1 1)
54     hex (2 3 6 5 12 13 16 15) (30 5 1) simpleGrading (1 1 1)
55     hex (3 4 7 6 13 14 17 16) (25 5 1) simpleGrading (1 1 1)
56     hex (5 6 9 8 15 16 19 18) (30 95 1) simpleGrading (1 1 1)
57 );
58
59 edges
60 (
61 );
62
63 patches
64 (
65     wall outerWall
66     (
67         (0 1 11 10)
68         (1 3 13 11)
69         (3 4 14 13)
70         (7 6 16 17)
71         (6 9 19 16)
72         (9 8 18 19)
73     )
74     symmetryPlane axis
75     (
76         (0 10 12 2)
77         (2 12 15 5)
78         (5 15 18 8)
79     )
80     patch nozzle
81     (
82         (4 7 17 14)
83     )
84     empty back
85     (
86         (0 2 3 1)
87         (2 5 6 3)
88         (3 6 7 4)
89         (5 8 9 6)
90     )
91     empty front
92     (
93         (10 11 13 12)
94         (12 13 16 15)
95         (13 14 17 16)
96         (15 16 19 18)
97     )
98 );
99
100 mergePatchPairs
101 (
102 );
103
104 // ***** //

```

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.

3.4.3 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:

$$c = \sqrt{\frac{1}{\psi}} = \sqrt{\frac{1}{4.54 \times 10^{-7}}} = 1483.2 \text{ m/s.} \quad (3.18)$$

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

$$Co = \frac{U \Delta t}{\Delta x} \quad (3.19)$$

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  /*-----*\
2  |=====|
3  | \      / | F i e l d           | OpenFOAM: The Open Source CFD Toolbox
4  |  \    /  | O p e r a t i o n   | Version: 1.3
5  |   \  /   | A n d                | Web:      http://www.openfoam.org
6  |    \/    | M a n i p u l a t i o n |
7  |-----*\
8
9  FoamFile
10 {
11     version      2.0;
12     format       ascii;
13
14     root         "";
15     case         "";
16     instance     "";
17     local        "";
18
19     class        dictionary;
20     object       controlDict;
21 }
22
23 // ***** //
24
25 application sonicLiquidFoam;
26
27 startFrom      startTime;
28
29 startTime      0;
30
31 stopAt         endTime;
32
33 endTime        0.0001;
34
35 deltaT         5e-07;
36
37 writeControl   timeStep;
38
39 writeInterval  20;
40
41 purgeWrite     0;
42
43 writeFormat    ascii;
44
45 writePrecision 6;
46
47 writeCompression compressed;
48
49 timeFormat     general;
50
51 timePrecision  6;
52
53 runtimeModifiable yes;
54
55
56 // ***** //

```

3.4.4 Running the case

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

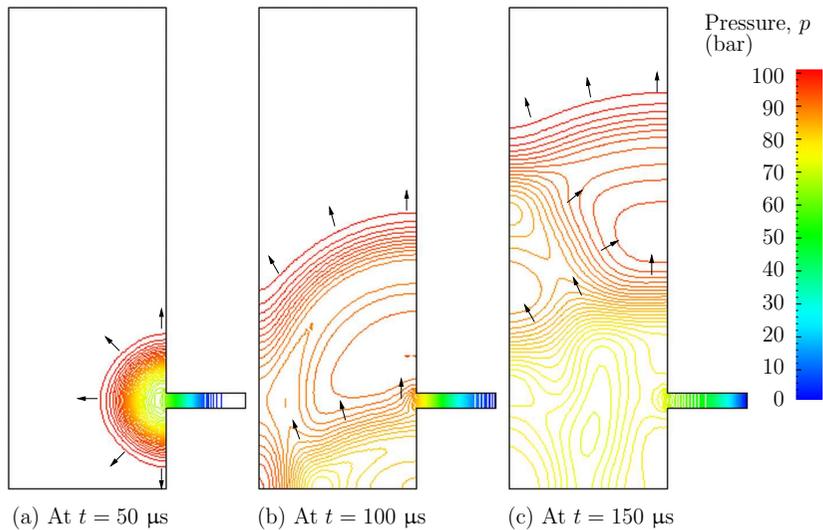


Figure 3.10: Propagation of pressure waves

propagate through the tank. In Figure 3.10, 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.

3.4.5 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 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 3.11.

3.5 Magnetohydrodynamic flow of a liquid

In this example we shall investigate an flow of an electrically-conducting liquid through a magnetic field. The problem is one belonging to the branch of fluid dynamics known as magnetohydrodynamics (MHD) that uses `mhdFoam`.

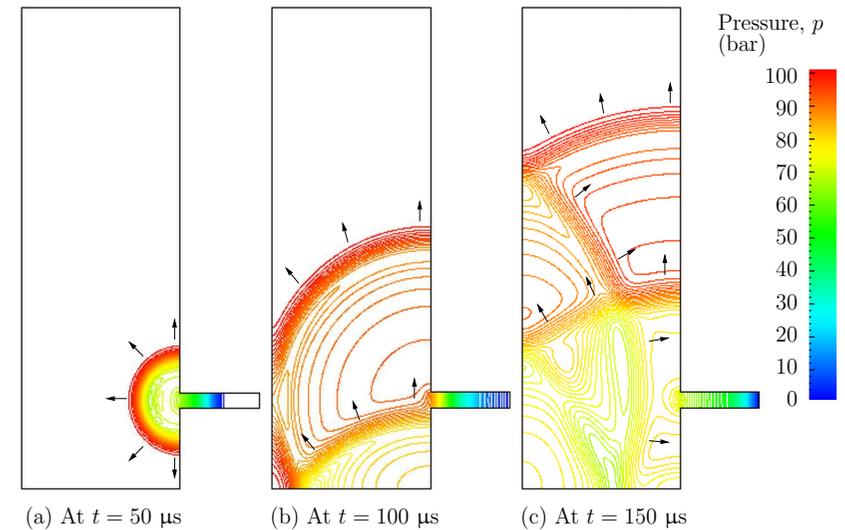


Figure 3.11: Propagation of pressure waves with refined mesh

3.5.1 Problem specification

The problem 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. 3.12.

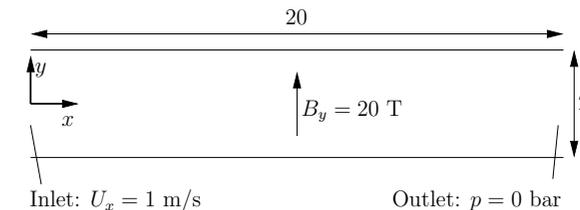


Figure 3.12: Geometry of the Hartmann problem

Governing equations

- Mass continuity for incompressible fluid

$$\nabla \cdot \mathbf{U} = 0 \quad (3.20)$$

- Momentum equation for incompressible fluid

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) + \nabla \cdot (2\mathbf{B}\Gamma_{\mathbf{B}}\mathbf{B}) + \nabla \cdot (\nu\mathbf{U}) + \nabla (\Gamma_{\mathbf{B}}\mathbf{B}:\mathbf{B}) = -\nabla p \quad (3.21)$$

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

3.5.3 Running the case

The user can run the case and view results in `dxFoam`. 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

$$M = BL\sqrt{\frac{\sigma}{\rho\nu}} \quad (3.29)$$

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

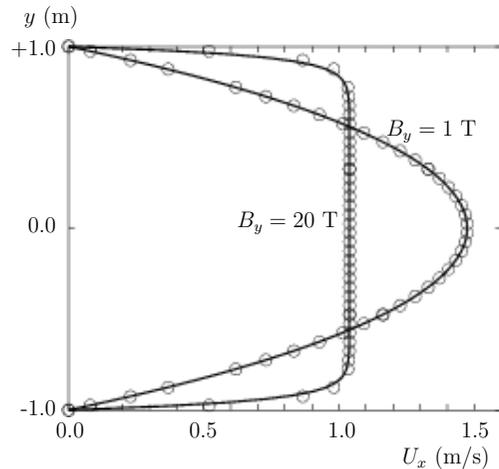


Figure 3.13: Velocity profile in the Hartmann problem for $B_y = 1$ T and $B_y = 20$ T.

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 3.13](#). To validate the code the analytical solution for the velocity profile U_x is superimposed in [Figure 3.13](#), given by:

$$\frac{U_x(y)}{U_x(0)} = \frac{\cosh M - \cosh M(y/L)}{\cosh M - 1} \quad (3.30)$$

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

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**

Symbols

* tensor member function, [P-25](#)

+ tensor member function, [P-25](#)

- tensor member function, [P-25](#)

/ tensor member function, [P-25](#)

/*...*/
C++ syntax, [U-76](#)

//
C++ syntax, [U-76](#)
OpenFOAM file syntax, [U-98](#)

include
C++ syntax, [U-70](#), [U-76](#)

&
tensor member function, [P-25](#)

&&
tensor member function, [P-25](#)

~
tensor member function, [P-25](#)

<LESmodel>Coeffs keyword, [U-178](#)

<delta>Coeffs keyword, [U-178](#)

<turbulenceModel>Coeffs keyword, [U-178](#)

0.000000e+00 directory, [U-98](#)

1-dimensional mesh, [U-136](#)

1D mesh, [U-136](#)

2-dimensional mesh, [U-136](#)

2D mesh, [U-136](#)

3D View button, [U-162](#)

3D view Properties
menu entry, [U-26](#), [U-162–U-164](#)

Numbers

0 directory, [U-98](#)

A

Accept button, [U-160](#)

access functions, [P-23](#)

Actor color button, [U-162](#)

adiabaticFlameT utility, [U-90](#)

adjustableRunTime

keyword entry, [U-59](#), [U-103](#)

adjustPhi tools, [U-91](#)

adjustTimeStep keyword, [U-59](#)

algebraic multi-grid, [U-113](#)

algorithms tools, [U-91](#)

allTime
menu entry, [U-124](#)

AMG
keyword entry, [U-112](#)

analytical solution, [P-45](#)

anisotropicFilter model, [U-94](#)

Annotate window panel, [U-26](#), [U-162](#)

APIfunctions model, [U-93](#)

applicationClass keyword, [U-103](#)

applications, [U-67](#)

arbitrarily unstructured, [P-31](#)

arc
keyword entry, [U-47](#), [U-146](#)

arc keyword, [U-145](#)

ascii
keyword entry, [U-104](#)

attachMesh utility, [U-87](#)

autoPatch utility, [U-87](#)

axes
right-handed, [U-144](#)
right-handed rectangular Cartesian, [P-15](#), [U-20](#)

axi-symmetric cases, [U-141](#), [U-150](#)

axi-symmetric mesh, [U-136](#)

B

background
process, [U-27](#), [U-79](#)

backward
keyword entry, [U-111](#)

Backward differencing, [P-39](#)

basicThermophysicalModels
library, [U-92](#)

BDCG
keyword entry, [U-112](#)

BICCG
keyword entry, [U-112](#)

binary
keyword entry, [U-104](#)

BirdCarreau model, [U-95](#)

blended differencing, [P-38](#)

block
expansion ratio, [U-147](#)

block keyword, [U-145](#)

blockMesh solver, [P-47](#)

blockMesh utility, [U-39](#), [U-86](#), [U-144](#)

blockMesh
menu entry, [U-22](#), [U-33](#)

blockMesh executable
vertex numbering, [U-146](#)

blockMeshDict
dictionary, [U-21](#), [U-22](#), [U-37](#), [U-47](#), [U-144](#), [U-150](#)

blocks keyword, [U-22](#), [U-146](#)

bound tools, [U-91](#)

boundaries, [U-136](#)

boundary, [U-136](#)

boundary
dictionary, [U-135](#), [U-144](#)

boundary condition
calculated, [U-142](#)
cyclic, [U-141](#)
directionMixed, [U-142](#)
empty, [P-64](#), [P-69](#), [U-20](#), [U-136](#), [U-140](#)
fixedGradient, [U-142](#)
fixedValue, [U-142](#)
fluxCorrectedVelocity, [U-143](#)
gammaContactAngle, [U-57](#)
inlet, [P-69](#)
inletOutlet, [U-143](#)
mixed, [U-142](#)
movingWallVelocity, [U-143](#)
outlet, [P-69](#)
outletInlet, [U-143](#)
partialSlip, [U-143](#)
patch, [U-140](#)
pressureDirectedInletVelocity, [U-143](#)
pressureInletVelocity, [U-143](#)
pressureOutlet, [P-64](#)
pressureTransmissive, [U-143](#)
processor, [U-141](#)
setup, [U-23](#)
slip, [U-143](#)
supersonicFreeStream, [U-143](#)
surfaceNormalFixedValue, [U-143](#)
symmetryPlane, [P-64](#), [U-140](#)
totalPressure, [U-143](#)
turbulentInlet, [U-143](#)
wall, [U-41](#)
wall, [P-64](#), [P-69](#), [U-140](#)
wallBuoyantPressure, [U-143](#)

wedge, [U-136](#), [U-141](#), [U-150](#)

zeroGradient, [U-142](#)

boundary conditions, [P-43](#)
Dirichlet, [P-43](#)
inlet, [P-44](#)
Neumann, [P-43](#)
no-slip impermeable wall, [P-44](#)
outlet, [P-44](#)
physical, [P-44](#)
symmetry plane, [P-44](#)

boundary type
empty, [U-127](#)
wall, [U-41](#)

boundaryField keyword, [U-102](#)

boundaryFoam solver, [U-84](#)

bounded
keyword entry, [U-109](#), [U-110](#)

boxToCell keyword, [U-57](#)

boxTurb utility, [U-86](#)

breaking of a dam, [U-55](#)

bubbleFoam solver, [U-85](#)

buoyantFoam solver, [U-86](#)

buoyantSimpleFoam solver, [U-86](#)

button
3D View, [U-162](#)
Accept, [U-160](#)
Actor color, [U-162](#)
Close Case, [U-31](#)
Compact, [U-127](#)
Delete, [U-161](#)
Display Orientation Axes, [U-162](#)
Info, [U-127](#)
My Jobs, [U-127](#)
Orientation Axes, [U-26](#)
Refresh Case Browser, [U-40](#)
Reset Range, [U-29](#)
Reset, [U-160](#)
Start Calculation Now, [U-27](#)
Start Calculation, [U-34](#)
Use parallel projection, [U-26](#), [U-162](#)
cont, [U-127](#)
endNow, [U-127](#)
end, [U-127](#)
kill, [U-127](#)
purge, [U-127](#)
read, [U-127](#)
status, [U-127](#)
suspend, [U-127](#)

C

C++ syntax
/*...*/, [U-76](#)
//, [U-76](#)
include, [U-70](#), [U-76](#)

calculated
 boundary condition, [U-142](#)
Camera window panel, [U-162](#)
Camera Controls window panel, [U-162](#)
Camera Orientation window panel, [U-162](#)
Case menu, [U-84](#)
case
 browser, [U-120](#)
 server, [U-126](#)
case keyword, [U-99](#)
case manager
 FoamX, [U-117](#)
Case Name text box, [U-122](#)
Case Root text box, [U-122](#)
caseRoots keyword, [U-19](#)
cases, [U-97](#)
cavity flow, [U-19](#)
CEI_ARCH
 environment variable, [U-168](#)
CEI_HOME
 environment variable, [U-168](#)
cell
 expansion ratio, [U-147](#)
cell class, [P-31](#)
cell
 keyword entry, [U-170](#)
cellDecompFiniteElement
 library, [U-91](#)
cellPoint
 keyword entry, [U-170](#)
cellPointFace
 keyword entry, [U-170](#)
cells
 dictionary, [U-135](#), [U-144](#)
cellSet utility, [U-87](#)
central differencing, [P-38](#)
cfdTools
 library, [U-91](#)
cfxToFoam utility, [U-87](#), [U-151](#)
cGamma keyword, [U-61](#)
channelOodles solver, [U-85](#)
checkMesh utility, [U-87](#), [U-152](#)
checkYPlus utility, [U-89](#)
chemistryModel
 library, [U-93](#)
chemistryModel model, [U-93](#)
chemistrySolver model, [U-93](#)
chemkinMixture model, [U-92](#), [U-176](#)
Class menu, [U-122](#)
class
 cell, [P-31](#)
 dimensionSet, [P-25](#), [P-32](#), [P-33](#)
 face, [P-31](#)
 finiteVolumeCalculus, [P-33](#)
 finiteVolumeMethod, [P-33](#)
 fvMesh, [P-31](#), [U-135](#)
 fvSchemes, [P-36](#)
 fvc, [P-36](#)
 fvm, [P-36](#)
 pointField, [P-31](#)
 polyBoundaryMesh, [P-31](#)
 polyMesh, [P-31](#), [U-133](#), [U-135](#)
 polyPatchList, [P-31](#)
 polyPatch, [P-31](#)
 scalarField, [P-29](#)
 scalar, [P-23](#)
 slice, [P-31](#)
 symmTensorField, [P-29](#)
 symmTensorThirdField, [P-29](#)
 tensorField, [P-29](#)
 tensorThirdField, [P-29](#)
 tensor, [P-23](#)
 vectorField, [P-29](#)
 vector, [P-23](#), [U-101](#)
 word, [P-25](#), [P-31](#)
class keyword, [U-99](#)
clockTime
 keyword entry, [U-103](#)
Close Case button, [U-31](#)
cloud keyword, [U-171](#)
cmptAv
 tensor member function, [P-25](#)
Co utility, [U-88](#)
cofactors
 tensor member function, [P-25](#)
coldEngineFoam solver, [U-85](#)
Color by menu, [U-162](#)
combustionThermophysicalModels
 library, [U-92](#)
comments, [U-76](#)
Compact button, [U-127](#)
compressed
 keyword entry, [U-104](#)
compressible tools, [U-91](#)
compressibleLESmodels
 library, [U-95](#)
compressibleTurbulenceModels
 library, [U-94](#)
constant directory, [U-98](#), [U-175](#)
constLaminarFlameSpeed model, [U-92](#)
constTransport model, [U-93](#), [U-176](#)
cont button, [U-127](#)
contactStressFoam solver, [U-86](#)
containers tools, [U-91](#)
continuum
 mechanics, [P-15](#)
control
 of time, [U-102](#)

controlDict
 dictionary, [P-66](#), [U-24](#), [U-33](#), [U-42](#), [U-51](#),
 [U-59](#), [U-98](#), [U-157](#)
controlDict file, [P-49](#)
convection, *see* divergence, [P-38](#)
convergence, [U-40](#)
convertToMeters keyword, [U-144](#), [U-145](#)
coordinate
 system, [P-15](#)
coordinate system, [U-20](#)
CORBA, [U-92](#), [U-117](#)
corrected
 keyword entry, [U-109](#), [U-110](#)
couplePatches utility, [U-87](#)
Courant number, [P-42](#), [U-25](#)
cpuTime
 keyword entry, [U-103](#)
Crank Nicholson
 temporal discretisation, [P-42](#)
CrankNicholson
 keyword entry, [U-111](#)
createPatch utility, [U-87](#)
cross product, *see* tensor, vector cross product
CrossPowerLaw
 keyword entry, [U-58](#)
CrossPowerLaw model, [U-95](#)
cubeRootVolDelta model, [U-94](#)
cubicCorrected
 keyword entry, [U-110](#)
cubicCorrection
 keyword entry, [U-108](#)
curl, [P-37](#)
curl
 fvc member function, [P-37](#)
curve keyword, [U-171](#)
cyclic
 boundary condition, [U-141](#)
cyclic
 keyword entry, [U-141](#)
cylinder
 flow around a, [P-45](#)

D

d2dt2
 fvc member function, [P-37](#)
 fvm member function, [P-37](#)
dam
 breaking of a, [U-55](#)
db tools, [U-91](#)
DCG
 keyword entry, [U-112](#)
ddt
 fvc member function, [P-37](#)
 fvm member function, [P-37](#)
DeardorffDiffStress model, [U-94](#), [U-95](#)
decomposePar utility, [U-80](#), [U-81](#), [U-90](#)
decomposeParDict
 dictionary, [U-80](#)
decomposition
 of field, [U-80](#)
 of mesh, [U-80](#)
decompression of a tank, [P-62](#)
defaultFieldValues keyword, [U-57](#)
deformedGeom utility, [U-87](#)
Delete button, [U-161](#)
delta keyword, [U-81](#), [U-178](#)
deltaT keyword, [U-103](#)
dependencies, [U-70](#)
dependency lists, [U-70](#)
det
 tensor member function, [P-25](#)
determinant, *see* tensor, determinant
dev
 tensor member function, [P-25](#)
diag
 tensor member function, [P-25](#)
Dictionaries dictionary tree, [U-128](#)
dictionary
 PISO, [U-25](#)
 blockMeshDict, [U-21](#), [U-22](#), [U-37](#), [U-47](#),
 [U-144](#), [U-150](#)
 boundary, [U-135](#), [U-144](#)
 cells, [U-135](#), [U-144](#)
 controlDict, [P-66](#), [U-24](#), [U-33](#), [U-42](#), [U-51](#),
 [U-59](#), [U-98](#), [U-157](#)
 decomposeParDict, [U-80](#)
 faces, [U-135](#), [U-144](#)
 fvSchemes, [U-60](#), [U-98](#), [U-105](#)
 fvSolution, [U-98](#), [U-111](#)
 mechanicalProperties, [U-50](#)
 points, [U-135](#), [U-144](#)
 thermalProperties, [U-50](#), [U-51](#)
 thermophysicalProperties, [U-175](#)
 transportProperties, [U-24](#), [U-40](#), [U-42](#)
 turbulenceProperties, [U-42](#), [U-178](#)
dictionary tree
 Dictionaries, [U-128](#)
 Fields, [U-23](#), [U-127](#)
 Mesh, [U-23](#)
 Patches, [U-23](#)
dieselEngineFoam solver, [U-85](#)
dieselMixture model, [U-92](#), [U-176](#)
dieselSpray
 library, [U-92](#)
diEthylEther model, [U-93](#)
differencing
 Backward, [P-39](#)
 blended, [P-38](#)

central, [P-38](#)
 Euler implicit, [P-39](#)
 Gamma, [P-38](#)
 MINMOD, [P-38](#)
 SUPERBEE, [P-38](#)
 upwind, [P-38](#)
 van Leer, [P-38](#)

dimension
 checking in OpenFOAM, [P-25](#)

dimensioned<Type> template class, [P-25](#)

dimensionedTypes tools, [U-91](#)

dimensions keyword, [U-102](#)

dimensionSet class, [P-25](#), [P-32](#), [P-33](#)

dimensionSet tools, [U-91](#)

diMethylEther model, [U-93](#)

direct numerical simulation, [U-60](#)

directionMixed
 boundary condition, [U-142](#)

directory
0.000000e+00, [U-98](#)
0, [U-98](#)
Make, [U-71](#)
constant, [U-98](#), [U-175](#)
fluentInterface, [U-165](#)
polyMesh, [U-98](#), [U-135](#)
processorN, [U-81](#)
run, [U-97](#)
system, [P-49](#), [U-98](#)
tutorials, [P-45](#), [U-19](#)

discretisation
 equation, [P-33](#)

Display window panel, [U-26](#), [U-29](#),
[U-160](#), [U-161](#)

Display Orientation Axes button, [U-162](#)

distance
 keyword entry, [U-171](#)

distributed keyword, [U-81](#), [U-83](#)

div
 fvc member function, [P-37](#)
 fvm member function, [P-37](#)

divergence, [P-37](#), [P-39](#)

divSchemes keyword, [U-105](#)

divU utility, [U-88](#)

dnsFoam solver, [U-85](#)

double inner product, *see* tensor,double inner
 product

dxFoamExec utility, [U-88](#)

dynamicMesh
 library, [U-91](#)

dynMixedSmagorinsky model, [U-94](#)

dynOneEqEddy model, [U-94](#), [U-95](#)

dynSmagorinsky model, [U-94](#)

E

edgeGrading keyword, [U-147](#)

edgeMesh
 library, [U-91](#)

edges keyword, [U-145](#)

electrostaticFoam solver, [U-86](#)

empty
 boundary condition, [P-64](#), [P-69](#), [U-20](#),
[U-136](#), [U-140](#)

empty boundary type, [U-127](#)

empty
 keyword entry, [U-141](#)

end button, [U-127](#)

endNow button, [U-127](#)

endTime keyword, [U-25](#), [U-103](#)

engine
 library, [U-92](#)

engineCompRatio utility, [U-89](#)

engineFoam solver, [U-85](#)

engineSwirl utility, [U-86](#)

ensight74FoamExec utility, [U-88](#), [U-168](#)

ensight76FoamExec utility, [U-88](#)

ENSIGHT7_INPUT
 environment variable, [U-168](#)

ENSIGHT7_READER
 environment variable, [U-168](#)

enstrophy utility, [U-89](#)

environment variable
 CEI.ARCH, [U-168](#)
 CEI.HOME, [U-168](#)
 ENSIGHT7_INPUT, [U-168](#)
 ENSIGHT7_READER, [U-168](#)
 FOAMX_PATH, [U-132](#)
 FOAMX_SYSTEM_CONFIG, [U-132](#)
 FOAMX_USER_CONFIG, [U-132](#)
 FOAM_RUN, [U-97](#), [U-132](#)
 JAVA_HOME, [U-132](#)
 WM.ARCH, [U-74](#)
 WM_COMPILER_BIN, [U-74](#)
 WM_COMPILER_DIR, [U-74](#)
 WM_COMPILER_LIB, [U-74](#)
 WM_COMPILER, [U-74](#)
 WM_COMPILE_OPTION, [U-74](#)
 WM_DIR, [U-74](#)
 WM_JAVAC_OPTION, [U-74](#)
 WM_LINK_LANGUAGE, [U-74](#)
 WM_MPLIB, [U-74](#)
 WM_OPTIONS, [U-74](#)
 WM_PROJECT_DIR, [U-74](#)
 WM_PROJECT_INST_DIR, [U-74](#)
 WM_PROJECT_LANGUAGE, [U-74](#)
 WM_PROJECT_USER_DIR, [U-74](#)
 WM_PROJECT_VERSION, [U-74](#)
 WM_PROJECT, [U-74](#)

WM.SHELL, [U-74](#)
 wmake, [U-73](#)

environmentalProperties file, [U-59](#)

equilibriumCO utility, [U-90](#)

equilibriumFlameT utility, [U-90](#)

errorEstimation
 library, [U-91](#)

estimateScalarError utility, [U-90](#)

Euler
 keyword entry, [U-111](#)

Euler implicit
 differencing, [P-39](#)
 temporal discretisation, [P-42](#)

examples
 decompression of a tank, [P-62](#)
 flow around a cylinder, [P-45](#)
 flow over backward step, [P-54](#)
 Hartmann problem, [P-67](#)
 supersonic flow over forward step, [P-59](#)

explicit
 temporal discretisation, [P-42](#)

exponential model, [U-93](#)

F

face class, [P-31](#)

face keyword, [U-171](#)

faceDecompFiniteElement
 library, [U-91](#)

faces
 dictionary, [U-135](#), [U-144](#)

faceSet utility, [U-87](#)

field
 U, [U-25](#)
 p, [U-25](#)
 decomposition, [U-80](#)

FieldField<Type> template class, [P-32](#)

Fields dictionary tree, [U-23](#), [U-127](#)

Fields window, [U-29](#)

fields, [P-29](#)
 mapping, [U-157](#)

fields tools, [U-91](#)

fields keyword, [U-170](#)

fieldToCellSet utility, [U-87](#)

Field<Type> template class, [P-29](#)

fieldValues keyword, [U-57](#)

file
FoamX.cfg, [U-131](#)
FoamXClient.cfg, [U-118](#), [U-131](#)
Make/files, [U-72](#)
controlDict, [P-49](#)
environmentalProperties, [U-59](#)
files, [U-71](#)
options, [U-71](#)
transportProperties, [U-58](#)

file format, [U-98](#)

files file, [U-71](#)

financialFoam solver, [U-86](#)

finite volume
 discretisation, [P-27](#)
 mesh, [P-31](#)

finiteVolume tools, [U-91](#)

finiteVolumeCalculus class, [P-33](#)

finiteVolumeMethod class, [P-33](#)

firstTime
 menu entry, [U-124](#)

firstTime keyword, [U-103](#)

fixed
 keyword entry, [U-104](#)

fixedGradient
 boundary condition, [U-142](#)

fixedValue
 boundary condition, [U-142](#)

flattenMesh utility, [U-87](#)

flow
 free surface, [U-55](#)
 laminar, [U-19](#)
 steady, turbulent, [P-54](#)
 supersonic, [P-59](#)
 turbulent, [U-19](#)

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

flow over backward step, [P-54](#)

fluentInterface directory, [U-165](#)

fluentMeshToFoam utility, [U-87](#), [U-151](#)

fluxCorrectedVelocity
 boundary condition, [U-143](#)

fluxRequired keyword, [U-105](#)

OpenFOAM
 cases, [U-97](#)

FOAM_RUN
 environment variable, [U-97](#), [U-132](#)

Foam Utilities menu, [U-22](#), [U-33](#)

foamConvert21To22 utility, [U-90](#)

foamCorrectVrt script/alias, [U-155](#)

foamDataToFluent utility, [U-88](#), [U-165](#)

foamDebugSwitches utility, [U-90](#)

FoamFile keyword, [U-99](#)

foamInfoExec utility, [U-90](#)

foamJob script/alias, [U-172](#)

foamLog script/alias, [U-172](#)

foamMeshToFluent utility, [U-87](#), [U-165](#)

foamToDX utility, [U-88](#)

foamToEnight utility, [U-88](#)

foamToFieldview utility, [U-88](#)

foamToFieldview9 utility, [U-88](#)

foamToVTK utility, [U-88](#)

foamUser
 library, [U-78](#)

FoamX

- case browser, [U-120](#)
- case manager, [U-117](#)
- case server, [U-126](#)
- OpenFOAM case manager, [U-117](#)
- host browser, [U-118](#)
- JAVA GUI, [U-119](#)
- name server, [U-118](#)
- FoamX utility, [U-86](#)
- FoamX.cfg* file, [U-131](#)
- FOAMX_PATH
 - environment variable, [U-132](#)
- FOAMX.SYSTEM_CONFIG
 - environment variable, [U-132](#)
- FOAMX.USER_CONFIG
 - environment variable, [U-132](#)
- FoamXClient.cfg* file, [U-118](#), [U-131](#)
- foreground
 - process, [U-27](#)
- format keyword, [U-99](#)
- fourth
 - keyword entry, [U-109](#), [U-110](#)
- fvc class, [P-36](#)
- fvc member function
 - curl*, [P-37](#)
 - d2dt2*, [P-37](#)
 - ddt*, [P-37](#)
 - div*, [P-37](#)
 - gGrad*, [P-37](#)
 - grad*, [P-37](#)
 - laplacian*, [P-37](#)
 - lsGrad*, [P-37](#)
 - snGrad*, [P-37](#)
 - snGradCorrection*, [P-37](#)
 - sqrGradGrad*, [P-37](#)
- fvm class, [P-36](#)
- fvm member function
 - d2dt2*, [P-37](#)
 - ddt*, [P-37](#)
 - div*, [P-37](#)
 - laplacian*, [P-37](#)
 - Su*, [P-37](#)
 - SuSp*, [P-37](#)
- fvmMatrix template class, [P-33](#)
- fvmMesh class, [P-31](#), [U-135](#)
- fvschemes*
 - dictionary, [U-60](#), [U-98](#), [U-105](#)
- fvschemes class, [P-36](#)
- fvschemes
 - menu entry, [U-52](#)
- fvsolution*
 - dictionary, [U-98](#), [U-111](#)

G

- gambitToFoam utility, [U-87](#), [U-151](#)

- Gamma
 - keyword entry, [U-108](#)
- Gamma differencing, [P-38](#)
- gammaContactAngle
 - boundary condition, [U-57](#)
- Gauss
 - keyword entry, [U-109](#)
- Gauss's theorem, [P-36](#)
- GaussSeidel
 - keyword entry, [U-112](#)
- General window panel, [U-162](#)
- general model, [U-93](#)
- general
 - keyword entry, [U-104](#)
- GeometricBoundaryField template class, [P-32](#)
- geometricField<Type> template class, [P-32](#)
- gGrad
 - fvc member function, [P-37](#)
- global tools, [U-91](#)
- gmshToFoam utility, [U-87](#)
- gnuplot
 - keyword entry, [U-104](#), [U-170](#)
- grad
 - fvc member function, [P-37](#)
- (Grad Grad) squared, [P-37](#)
- gradient, [P-37](#), [P-40](#)
 - Gauss scheme, [P-40](#)
 - Gauss's theorem, [U-52](#)
 - least square fit, [U-52](#)
 - least squares method, [P-40](#), [U-52](#)
 - surface normal, [P-40](#)
- gradSchemes keyword, [U-105](#)
- graphFormat keyword, [U-104](#)
- Gstream
 - library, [U-92](#)
- guldersLaminarFlameSpeed model, [U-93](#)

H

- hConstThermo model, [U-93](#), [U-175](#)
- hhuMixtureThermo model, [U-92](#), [U-176](#)
- hierarchical
 - keyword entry, [U-80](#), [U-81](#)
- hMixtureThermo model, [U-92](#), [U-176](#)
- homogeneousMixture model, [U-92](#), [U-176](#)
- host, [U-20](#)
 - browser, [U-118](#)
- hThermo model, [U-92](#), [U-176](#)

I

- I
 - tensor member function, [P-25](#)
- ICCG
 - keyword entry, [U-112](#)
- icoErrorEstimate utility, [U-90](#)

- icoFoam solver, [U-19](#), [U-24](#), [U-25](#), [U-27](#), [U-84](#)
- icoFoamAutoMotion solver, [U-84](#)
- icoMomentError utility, [U-90](#)
- icoTopoFoam solver, [U-85](#)
- ideasToFoam utility, [U-87](#), [U-151](#)
- identities, *see* tensor, identities
- identity, *see* tensor, identity
- incompressible tools, [U-91](#)
- incompressibleLESmodels
 - library, [U-94](#)
- incompressiblePostProcessing
 - library, [U-91](#)
- incompressibleTransportModels
 - library, [P-55](#), [U-95](#)
- incompressibleTurbulenceModels
 - library, [P-55](#), [U-93](#)
- index
 - notation, [P-16](#), [P-17](#)
- Info button, [U-127](#)
- Information window panel, [U-160](#)
- inhomogeneousMixture model, [U-92](#), [U-176](#)
- inlet
 - boundary condition, [P-69](#)
- inletOutlet
 - boundary condition, [U-143](#)
- inner product, *see* tensor, inner product
- insideCells utility, [U-87](#)
- instance keyword, [U-99](#)
- interFoam solver, [U-85](#)
- internalField keyword, [U-102](#), [U-127](#)
- interpolationScheme keyword, [U-170](#)
- interpolations tools, [U-91](#)
- interpolationSchemes keyword, [U-105](#)
- inv
 - tensor member function, [P-25](#)
- isoOctane model, [U-93](#)

J

- janafThermo model, [U-93](#), [U-175](#)
- JAVA_HOME
 - environment variable, [U-132](#)
- jplot
 - keyword entry, [U-104](#), [U-170](#)

K

- kappa keyword, [U-178](#)
- kEpsilon model, [U-93](#), [U-94](#)
- keyword
 - FoamFile, [U-99](#)
 - LESmodel, [U-178](#)
 - adjustTimeStep, [U-59](#)
 - applicationClass, [U-103](#)
 - arc, [U-145](#)
 - blocks, [U-22](#), [U-146](#)

- block, [U-145](#)
- boundaryField, [U-102](#)
- boxToCell, [U-57](#)
- cGamma, [U-61](#)
- caseRoots, [U-19](#)
- case, [U-99](#)
- class, [U-99](#)
- cloud, [U-171](#)
- convertToMeters, [U-144](#), [U-145](#)
- curve, [U-171](#)
- defaultFieldValues, [U-57](#)
- deltaT, [U-103](#)
- delta, [U-81](#), [U-178](#)
- dimensions, [U-102](#)
- distributed, [U-81](#), [U-83](#)
- divSchemes, [U-105](#)
- edgeGrading, [U-147](#)
- edges, [U-145](#)
- endTime, [U-25](#), [U-103](#)
- face, [U-171](#)
- fieldValues, [U-57](#)
- fields, [U-170](#)
- firstTime, [U-103](#)
- fluxRequired, [U-105](#)
- format, [U-99](#)
- gradSchemes, [U-105](#)
- graphFormat, [U-104](#)
- instance, [U-99](#)
- internalField, [U-102](#), [U-127](#)
- interpolationSchemes, [U-105](#)
- interpolationScheme, [U-170](#)
- kappa, [U-178](#)
- laplacianSchemes, [U-105](#)
- latestTime, [U-40](#)
- leastSquares, [U-52](#)
- local, [U-99](#)
- manualCoeffs, [U-81](#)
- maxCo, [U-59](#)
- maxDeltaT, [U-59](#)
- method, [U-81](#)
- metisCoeffs, [U-81](#)
- midPointAndFace, [U-171](#)
- midPoint, [U-171](#)
- nFaces, [U-135](#)
- nGammaSubCycles, [U-61](#)
- numberOfSubdomains, [U-81](#)
- n, [U-81](#)
- object, [U-99](#)
- order, [U-81](#)
- outputFormat, [U-170](#)
- pRefCell, [U-25](#), [U-115](#)
- pRefValue, [U-25](#), [U-115](#)
- patchMap, [U-157](#)
- patches, [U-145](#), [U-147](#)

pdRefCell, U-115
 pdRefValue, U-115
 physicalType, U-135, U-139
 processorWeights, U-81
 purgeWrite, U-103
 refGradient, U-142
 referenceLevel, U-102, U-127
 regions, U-57
 roots, U-81, U-83
 root, U-99
 runtimeModifiable, U-104
 sampleSets, U-170
 simpleGrading, U-147
 snGradSchemes, U-105
 solvers, U-112
 spline, U-145
 startFace, U-135
 startFrom, U-24, U-103
 startTime, U-24, U-103
 stopAt, U-103
 thermoType, U-175
 timeFormat, U-104
 timePrecision, U-104
 timeScheme, U-105
 topoSetSource, U-57
 turbulenceModel, U-178
 turbulence, U-178
 type, U-139
 uniform, U-171
 valueFraction, U-142
 value, U-142
 version, U-99
 vertices, U-22, U-145
 wallFunctionCoeffs, U-178
 writeCompression, U-104
 writeControl, U-25, U-59, U-103
 writeFormat, U-54, U-104
 writeInterval, U-25, U-34, U-103
 writePrecision, U-104
 <LESmodel>Coeffs, U-178
 <delta>Coeffs, U-178
 <turbulenceModel>Coeffs, U-178

keyword entry

- AMG, U-112
- BDCG, U-112
- BICCG, U-112
- CrankNicholson, U-111
- CrossPowerLaw, U-58
- DCG, U-112
- Euler, U-111
- Gamma, U-108
- GaussSeidel, U-112
- Gauss, U-109
- ICCG, U-112

MUSCL, U-108
 Newtonian, U-58
 QUICK, U-108, U-110
 SFCD, U-108, U-110
 UMIST, U-107
 adjustableRunTime, U-59, U-103
 arc, U-47, U-146
 ascii, U-104
 backward, U-111
 binary, U-104
 bounded, U-109, U-110
 cellPointFace, U-170
 cellPoint, U-170
 cell, U-170
 clockTime, U-103
 compressed, U-104
 corrected, U-109, U-110
 cpuTime, U-103
 cubicCorrected, U-110
 cubicCorrection, U-108
 cyclic, U-141
 distance, U-171
 empty, U-141
 fixed, U-104
 fourth, U-109, U-110
 general, U-104
 gnuplot, U-104, U-170
 hierarchical, U-80, U-81
 jplot, U-104, U-170
 latestTime, U-103
 leastSquares, U-109
 limitedCubic, U-108
 limitedLinear, U-108
 limited, U-109, U-110
 linearUpwind, U-108, U-110
 linear, U-108, U-110
 line, U-146
 manual, U-80, U-81
 metis, U-80, U-81
 midPoint, U-108
 nextWrite, U-103
 noWriteNow, U-103
 none, U-106
 patch, U-141
 polyLine, U-146
 polySpline, U-146
 processor, U-141
 raw, U-104, U-170
 runTime, U-34, U-103
 scientific, U-104
 simpleSpline, U-146
 simple, U-80, U-81
 skewLinear, U-108, U-110
 startTime, U-24, U-103

steadyState, U-111
 symmetryPlane, U-141
 timeStep, U-25, U-34, U-103
 uncompressed, U-104
 uncorrected, U-109, U-110
 upwind, U-108, U-110
 vanLeer, U-108
 wall, U-141
 wedge, U-141
 writeControl, U-103
 writeNow, U-103
 xmgr, U-104, U-170
 xyz, U-171
 x, U-171
 y, U-171
 z, U-171

kill button, U-127
 kivaToFoam utility, U-87
 Kronecker delta, P-20

L

lagrangian
 library, U-92

LAM
 message passing interface, U-81
 MPI, U-81

Lambda2 utility, U-89

LamBremhorstKE model, U-94

laminar model, U-93, U-94

laminarFlameSpeedModels
 library, U-92

laplaceFilter model, U-94

Laplacian, P-38

laplacian, P-37

laplacian
 fvc member function, P-37
 fvm member function, P-37

laplacianFoam solver, U-84

laplacianSchemes keyword, U-105

latestTime
 keyword entry, U-103
 menu entry, U-124

latestTime keyword, U-40

LaunderGibsonRSTM model, U-94

LaunderSharmaKE model, U-94

leastSquares
 keyword entry, U-109

leastSquares keyword, U-52

LESdeltas
 library, U-94

LESfilters
 library, U-94

lesInterFoam solver, U-85

LESmodel keyword, U-178

libraries, U-67

library
 Gstream, U-92
 LESdeltas, U-94
 LESfilters, U-94
 ODE, U-91
 OpenFOAM, U-91
 PVFoamReader, U-159
 basicThermophysicalModels, U-92
 cellDecompFiniteElement, U-91
 cfdTools, U-91
 chemistryModel, U-93
 combustionThermophysicalModels, U-92
 compressibleLESmodels, U-95
 compressibleTurbulenceModels, U-94
 dieselSpray, U-92
 dynamicMesh, U-91
 edgeMesh, U-91
 engine, U-92
 errorEstimation, U-91
 faceDecompFiniteElement, U-91
 foamUser, U-78
 incompressibleLESmodels, U-94
 incompressiblePostProcessing, U-91
 incompressibleTransportModels, P-55, U-95
 incompressibleTurbulenceModels, P-55, U-93

lagrangian, U-92

laminarFlameSpeedModels, U-92

liquids, U-93

meshTools, U-91

mico-2.3.11, U-92

mpich-1.2.4, U-92

pdf, U-93

primitive, P-23

randomProcesses, U-92

sampling, U-91

shapeMeshTools, U-91

specie, U-93

thermophysicalFunctions, U-93

thermophysical, U-175

triSurface, U-92

vtkFoam, U-159

zlib-1.2.1, U-92

lid-driven cavity flow, U-19

LienCubicKE model, U-94

LienCubicKELowRE model, U-94

LienLeschzinerLowRE model, U-94

liftDrag utility, U-89

limited
 keyword entry, U-109, U-110

limitedCubic
 keyword entry, U-108

limitedLinear

keyword entry, [U-108](#)

line
keyword entry, [U-146](#)

linear
keyword entry, [U-108](#), [U-110](#)

linearUpwind
keyword entry, [U-108](#), [U-110](#)

liquid
electrically-conducting, [P-67](#)

liquids
library, [U-93](#)

lists, [P-29](#)

List<Type> template class, [P-29](#)

local keyword, [U-99](#)

locDynOneEqEddy model, [U-94](#)

Lower and Upper Times text box, [U-161](#)

lowReOneEqEddy model, [U-95](#)

LRDDiffStress model, [U-94](#)

LRR model, [U-94](#)

lsGrad
fvc member function, [P-37](#)

M

Mach utility, [U-89](#)

mag
tensor member function, [P-25](#)

magGradU utility, [U-89](#)

magnetohydrodynamics, [P-67](#)

magSqr
tensor member function, [P-25](#)

magU utility, [U-35](#), [U-89](#)

Make directory, [U-71](#)

make script/alias, [U-69](#)

Make/files file, [U-72](#)

makePolyMesh utility, [U-86](#)

manual
keyword entry, [U-80](#), [U-81](#)

manualCoeffs keyword, [U-81](#)

mapFields utility, [U-33](#), [U-39](#), [U-42](#), [U-54](#), [U-86](#), [U-157](#)

mapFields
menu entry, [U-33](#)

mapping
fields, [U-157](#)

matrices tools, [U-91](#)

max
tensor member function, [P-25](#)

maxCo keyword, [U-59](#)

maxDeltaT keyword, [U-59](#)

mechanicalProperties
dictionary, [U-50](#)

menu
Case, [U-84](#)
Class, [U-122](#)

Color by, [U-162](#)

Foam Utilities, [U-22](#), [U-33](#)

Mesh, [U-49](#)

View, [U-29](#), [U-162](#)

menu entry
3D view Properties, [U-26](#), [U-162–U-164](#)
Property, [U-162](#)
Read Mesh&Fields, [U-23](#), [U-44](#), [U-49](#)
Refresh Case Browser, [U-40](#)
Source, [U-29](#), [U-162](#)
Wireframe, [U-162](#)

allTime, [U-124](#)

blockMesh, [U-22](#), [U-33](#)

firstTime, [U-124](#)

fvSchemes, [U-52](#)

latestTime, [U-124](#)

mapFields, [U-33](#)

noTime, [U-124](#)

preProcessing, [U-33](#)

sample, [U-53](#)

mergeMeshes utility, [U-87](#)

Mesh dictionary tree, [U-23](#)

Mesh menu, [U-49](#)

mesh
1-dimensional, [U-136](#)
1D, [U-136](#)
2-dimensional, [U-136](#)
2D, [U-136](#)
axi-symmetric, [U-136](#)
basic, [P-31](#)
block structured, [U-144](#)
decomposition, [U-80](#)
description, [U-133](#)
finite volume, [P-31](#)
generation, [U-144](#)
grading, [U-144](#), [U-147](#)
grading, example of, [P-54](#)
non-orthogonal, [P-45](#)
refinement, [P-63](#)
resolution, [U-31](#)
specification, [U-133](#)
validity constraints, [U-133](#)

meshes tools, [U-91](#)

meshTools
library, [U-91](#)

message passing interface
LAM, [U-81](#)
MPICH, [U-181](#)

method keyword, [U-81](#)

metis
keyword entry, [U-80](#), [U-81](#)

metisCoeffs keyword, [U-81](#)

mhdFoam solver, [P-69](#), [U-86](#)

mico-2.3.11

library, [U-92](#)

midPoint
keyword entry, [U-108](#)

midPoint keyword, [U-171](#)

midPointAndFace keyword, [U-171](#)

min
tensor member function, [P-25](#)

MINMOD differencing, [P-38](#)

mirrorMesh utility, [U-87](#)

mixed
boundary condition, [U-142](#)

mixedSmagorinsky model, [U-94](#)

mixtureAdiabaticFlameT utility, [U-90](#)

model
APIfunctions, [U-93](#)
BirdCarreau, [U-95](#)
CrossPowerLaw, [U-95](#)
DeardorffDiffStress, [U-94](#), [U-95](#)
LRDDiffStress, [U-94](#)
LRR, [U-94](#)
LamBremhorstKE, [U-94](#)
LaunderGibsonRSTM, [U-94](#)
LaunderSharmaKE, [U-94](#)
LienCubicKELowRE, [U-94](#)
LienCubicKE, [U-94](#)
LienLeschzinerLowRE, [U-94](#)
NSRDSfunctions, [U-93](#)
Newtonian, [U-95](#)
NonlinearKESHih, [U-94](#)
PrandtlDelta, [U-94](#)
QZeta, [U-94](#)
RNGkEpsilon, [U-94](#)
RosinRammler, [U-93](#)
Smagorinsky2, [U-94](#)
Smagorinsky, [U-94](#), [U-95](#)
SpalartAllmaras, [U-94](#)
anisotropicFilter, [U-94](#)
chemistryModel, [U-93](#)
chemistrySolver, [U-93](#)
chemkinMixture, [U-92](#), [U-176](#)
constLaminarFlameSpeed, [U-92](#)
constTransport, [U-93](#), [U-176](#)
cubeRootVolDelta, [U-94](#)
diEthylEther, [U-93](#)
diMethylEther, [U-93](#)
dieselMixture, [U-92](#), [U-176](#)
dynMixedSmagorinsky, [U-94](#)
dynOneEqEddy, [U-94](#), [U-95](#)
dynSmagorinsky, [U-94](#)
exponential, [U-93](#)
general, [U-93](#)
guldersLaminarFlameSpeed, [U-93](#)
hConstThermo, [U-93](#), [U-175](#)
hMixtureThermo, [U-92](#), [U-176](#)

hThermo, [U-92](#), [U-176](#)

hhuMixtureThermo, [U-92](#), [U-176](#)

homogeneousMixture, [U-92](#), [U-176](#)

inhomogeneousMixture, [U-92](#), [U-176](#)

isoOctane, [U-93](#)

janafThermo, [U-93](#), [U-175](#)

kEpsilon, [U-93](#), [U-94](#)

laminar, [U-93](#), [U-94](#)

laplaceFilter, [U-94](#)

locDynOneEqEddy, [U-94](#)

lowReOneEqEddy, [U-95](#)

mixedSmagorinsky, [U-94](#)

multiComponentMixture, [U-92](#), [U-176](#)

nDecane, [U-93](#)

nDodecane, [U-93](#)

nHeptane, [U-93](#)

nOctane, [U-93](#)

normal, [U-93](#)

oneEqEddy, [U-94](#), [U-95](#)

perfectGas, [U-93](#), [U-175](#)

pureMixture, [U-92](#), [U-176](#)

scaleSimilarity, [U-94](#)

simpleFilter, [U-94](#)

smoothDelta, [U-94](#)

specieThermo, [U-93](#), [U-175](#)

spectEddyVisc, [U-94](#)

sutherlandTransport, [U-93](#), [U-176](#)

uniform, [U-93](#)

veryInhomogeneousMixture, [U-92](#), [U-176](#)

water, [U-93](#)

momentScalarError utility, [U-90](#)

moveEngineMesh utility, [U-87](#)

moveMesh utility, [U-87](#)

movingWallVelocity
boundary condition, [U-143](#)

MPI
LAM, [U-81](#)
MPICH, [U-181](#)

MPICH
message passing interface, [U-181](#)
MPI, [U-181](#)

mpich-1.2.4
library, [U-92](#)

mshToFoam utility, [U-87](#)

multi-grid
algebraic, [U-113](#)

multiComponentMixture model, [U-92](#), [U-176](#)

MUSCL
keyword entry, [U-108](#)

My Jobs button, [U-127](#)

N

n keyword, [U-81](#)

nabla

operator, [P-27](#)
 name
 server, [U-118](#)
 nDecane model, [U-93](#)
 nDodecane model, [U-93](#)
 Newtonian
 keyword entry, [U-58](#)
 Newtonian model, [U-95](#)
 nextWrite
 keyword entry, [U-103](#)
 nFaces keyword, [U-135](#)
 nGammaSubCycles keyword, [U-61](#)
 nHeptane model, [U-93](#)
 nOctane model, [U-93](#)
 non-orthogonal mesh, [P-45](#)
 none
 keyword entry, [U-106](#)
 NonlinearKEShiv model, [U-94](#)
 nonNewtonianIcoFoam solver, [U-85](#)
 normal model, [U-93](#)
 noTime
 menu entry, [U-124](#)
 noWriteNow
 keyword entry, [U-103](#)
 NSRDSfunctions model, [U-93](#)
 numberOfSubdomains keyword, [U-81](#)
 numerical diffusion, [U-60](#)

O

object keyword, [U-99](#)
 objToVTK utility, [U-87](#)
 ODE
 library, [U-91](#)
 oneEqEddy model, [U-94](#), [U-95](#)
 oodles solver, [U-85](#)
 Opacity text box, [U-162](#)
 OpenFOAM
 applications, [U-67](#)
 file format, [U-98](#)
 libraries, [U-67](#)
 OpenFOAM
 library, [U-91](#)
 OpenFOAM file syntax
 //, [U-98](#)
 operator
 scalar, [P-28](#)
 vector, [P-27](#)
 options file, [U-71](#)
 order keyword, [U-81](#)
 Orientation Axes button, [U-26](#)
 outer product, *see* tensor, outer product
 outlet
 boundary condition, [P-69](#)
 outletInlet

boundary condition, [U-143](#)
 outputFormat keyword, [U-170](#)

P

p field, [U-25](#)
 paraFoam, [U-26](#), [U-159](#)
 parallel
 running, [U-79](#)
 Parameters window panel, [U-29](#), [U-160](#), [U-161](#)
 partialSlip
 boundary condition, [U-143](#)
 patch
 boundary condition, [U-140](#)
 patch
 keyword entry, [U-141](#)
 Patches dictionary tree, [U-23](#)
 patches keyword, [U-145](#), [U-147](#)
 patchMap keyword, [U-157](#)
 pdf
 library, [U-93](#)
 pdRefCell keyword, [U-115](#)
 pdRefValue keyword, [U-115](#)
 Pe utility, [U-89](#)
 perfectGas model, [U-93](#), [U-175](#)
 permutation symbol, [P-19](#)
 physicalType keyword, [U-135](#), [U-139](#)
 PISO
 dictionary, [U-25](#)
 pointField class, [P-31](#)
 pointField<Type> template class, [P-33](#)
 points
 dictionary, [U-135](#), [U-144](#)
 pointSet utility, [U-87](#)
 polyBoundaryMesh class, [P-31](#)
 polyLine
 keyword entry, [U-146](#)
 polyMesh directory, [U-98](#), [U-135](#)
 polyMesh class, [P-31](#), [U-133](#), [U-135](#)
 polyPatch class, [P-31](#)
 polyPatchList class, [P-31](#)
 polySpline
 keyword entry, [U-146](#)
 post-processing, [U-159](#)
 post-processing
 paraFoam, [U-159](#)
 postChannel utility, [U-89](#)
 potentialFoam solver, [P-46](#), [U-84](#)
 pow
 tensor member function, [P-25](#)
 PrandtlDelta model, [U-94](#)
 pRefCell keyword, [U-25](#), [U-115](#)
 pRefValue keyword, [U-25](#), [U-115](#)
 preProcessing
 menu entry, [U-33](#)

pressure waves
 in liquids, [P-63](#)
 pressureDirectedInletVelocity
 boundary condition, [U-143](#)
 pressureInletVelocity
 boundary condition, [U-143](#)
 pressureOutlet
 boundary condition, [P-64](#)
 pressureTransmissive
 boundary condition, [U-143](#)
 primitive
 library, [P-23](#)
 primitives tools, [U-91](#)
 process
 background, [U-27](#), [U-79](#)
 foreground, [U-27](#)
 processor
 boundary condition, [U-141](#)
 processor
 keyword entry, [U-141](#)
 processorN directory, [U-81](#)
 processorWeights keyword, [U-81](#)
 Property
 menu entry, [U-162](#)
 ptot utility, [U-89](#)
 pureMixture model, [U-92](#), [U-176](#)
 purge button, [U-127](#)
 purgeWrite keyword, [U-103](#)
 PVFoamReader
 library, [U-159](#)

Q

Q utility, [U-89](#)
 QUICK
 keyword entry, [U-108](#), [U-110](#)
 QZeta model, [U-94](#)

R

R utility, [U-89](#)
 randomProcesses
 library, [U-92](#)
 rasInterFoam solver, [U-85](#)
 raw
 keyword entry, [U-104](#), [U-170](#)
 Rcomponents utility, [U-89](#)
 read button, [U-127](#)
 Read Mesh&Fields
 menu entry, [U-23](#), [U-44](#), [U-49](#)
 reconstructPar utility, [U-84](#), [U-90](#)
 referenceLevel keyword, [U-102](#), [U-127](#)
 refGradient keyword, [U-142](#)
 refineMesh utility, [U-87](#)
 refineShapeMesh utility, [U-87](#)
 Refresh Case Browser button, [U-40](#)

Refresh Case Browser
 menu entry, [U-40](#)
 Region window, [U-29](#)
 regions keyword, [U-57](#)
 relative tolerance, [U-113](#)
 renumberMesh utility, [U-88](#)
 Reset button, [U-160](#)
 Reset Range button, [U-29](#)
 restart, [U-40](#)
 Reynolds number, [U-19](#), [U-24](#)
 rhoSonicFoam solver, [U-85](#)
 rhoSonicFoam solver, [U-85](#)
 rmdepall script/alias, [U-74](#)
 RNgkEpsilon model, [U-94](#)
 root keyword, [U-99](#)
 roots keyword, [U-81](#), [U-83](#)
 RosinRammler model, [U-93](#)
 run
 parallel, [U-79](#)
 run directory, [U-97](#)
 runFoamX script/alias, [U-117](#)–[U-119](#)
 runFoamXHB script/alias, [U-117](#), [U-118](#)
 runTime
 keyword entry, [U-34](#), [U-103](#)
 runTimeModifiable keyword, [U-104](#)

S

sammToFoam utility, [U-87](#)
 sample utility, [U-90](#), [U-169](#)
 sample
 menu entry, [U-53](#)
 sampleSets keyword, [U-170](#)
 sampleSurface utility, [U-90](#)
 sampling
 library, [U-91](#)
 scalar, [P-16](#)
 operator, [P-28](#)
 scalar class, [P-23](#)
 scalarField class, [P-29](#)
 scalarTransportFoam solver, [U-84](#)
 scale
 tensor member function, [P-25](#)
 scalePoints utility, [U-88](#), [U-154](#)
 scaleSimilarity model, [U-94](#)
 scientific
 keyword entry, [U-104](#)
 script/alias
 foamCorrectVrt, [U-155](#)
 foamJob, [U-172](#)
 foamLog, [U-172](#)
 make, [U-69](#)
 rmdepall, [U-74](#)
 runFoamXHB, [U-117](#), [U-118](#)
 runFoamX, [U-117](#)–[U-119](#)

- wclean, U-73
- wmake, U-69
- second time derivative, P-37
- Seed window, U-163
- Selection Window window, U-26, U-160
- setFields utility, U-57, U-58
- settlingFoam solver, U-85
- SFCD
 - keyword entry, U-108, U-110
- shape, U-146
- shapeMeshTools
 - library, U-91
- simple
 - keyword entry, U-80, U-81
- simpleFilter model, U-94
- simpleFoam solver, P-55, U-85
- simpleGrading keyword, U-147
- simpleSpline
 - keyword entry, U-146
- skew
 - tensor member function, P-25
- skewLinear
 - keyword entry, U-108, U-110
- SI utility, U-90
- slice class, P-31
- slip
 - boundary condition, U-143
- Smagorinsky model, U-94, U-95
- Smagorinsky2 model, U-94
- smapToFoam utility, U-88
- smoothDelta model, U-94
- snGrad
 - fvc member function, P-37
- snGradCorrection
 - fvc member function, P-37
- snGradSchemes keyword, U-105
- solver
 - XiFoam, U-85
 - Xoodles, U-85
 - blockMesh, P-47
 - boundaryFoam, U-84
 - bubbleFoam, U-85
 - buoyantFoam, U-86
 - buoyantSimpleFoam, U-86
 - channelOodles, U-85
 - coldEngineFoam, U-85
 - contactStressFoam, U-86
 - dieselEngineFoam, U-85
 - dnsFoam, U-85
 - electrostaticFoam, U-86
 - engineFoam, U-85
 - financialFoam, U-86
 - icoFoamAutoMotion, U-84
 - icoFoam, U-19, U-24, U-25, U-27, U-84
 - icoTopoFoam, U-85
 - interFoam, U-85
 - laplacianFoam, U-84
 - lesInterFoam, U-85
 - mhdFoam, P-69, U-86
 - nonNewtonianIcoFoam, U-85
 - oodles, U-85
 - potentialFoam, P-46, U-84
 - rasInterFoam, U-85
 - rhoSonicFoam, U-85
 - rhoPsonicFoam, U-85
 - scalarTransportFoam, U-84
 - settlingFoam, U-85
 - simpleFoam, P-55, U-85
 - sonicFoamAutoMotion, U-85
 - sonicFoam, P-60, U-85
 - sonicLiquidFoam, P-64, U-85
 - sonicTurbFoam, U-85
 - stressFemFoam, U-86
 - stressedFoam, U-50, U-86
 - turbFoam, U-19, U-85
- solver relative tolerance, U-113
- solver tolerance, U-113
- solvers keyword, U-112
- sonicFoam solver, P-60, U-85
- sonicFoamAutoMotion solver, U-85
- sonicLiquidFoam solver, P-64, U-85
- sonicTurbFoam solver, U-85
- Source
 - menu entry, U-29, U-162
- source, P-37
- SpalartAllmaras model, U-94
- specie
 - library, U-93
- specieThermo model, U-93, U-175
- spectEddyVisc model, U-94
- spline keyword, U-145
- splitMesh utility, U-88
- splitMeshRegions utility, U-88
- sqr
 - tensor member function, P-25
- sqrGradGrad
 - fvc member function, P-37
- Standard Views window panel, U-162
- Start Calculation button, U-34
- Start Calculation Now button, U-27
- startFace keyword, U-135
- startFrom keyword, U-24, U-103
- starToFoam utility, U-87, U-151
- startTime
 - keyword entry, U-24, U-103
- startTime keyword, U-24, U-103
- status button, U-127
- steady flow

- turbulent, P-54
- steadyState
 - keyword entry, U-111
- stitchMesh utility, U-88
- stopAt keyword, U-103
- Stored Camera Position window panel, U-162
- streamFunction utility, U-89
- stress analysis of plate with hole, U-44
- stressComponents utility, U-89
- stressedFoam solver, U-50, U-86
- stressFemFoam solver, U-86
- Su
 - fvm member function, P-37
- subsetMesh utility, U-88
- summation convention, P-17
- SUPERBEE differencing, P-38
- supersonic flow, P-59
- supersonic flow over forward step, P-59
- supersonicFreeStream
 - boundary condition, U-143
- surfaceField<Type> template class, P-33
- surfaceNormalFixedValue
 - boundary condition, U-143
- SuSp
 - fvm member function, P-37
- suspend button, U-127
- sutherlandTransport model, U-93, U-176
- symm
 - tensor member function, P-25
- symmetryPlane
 - boundary condition, P-64, U-140
- symmetryPlane
 - keyword entry, U-141
- symmTensorField class, P-29
- symmTensorThirdField class, P-29
- system directory, P-49, U-98

T

- T()
 - tensor member function, P-25
- template class
 - GeometricBoundaryField, P-32
 - fvMatrix, P-33
 - dimensioned<Type>, P-25
 - FieldField<Type>, P-32
 - Field<Type>, P-29
 - geometricField<Type>, P-32
 - List<Type>, P-29
 - pointField<Type>, P-33
 - surfaceField<Type>, P-33
 - volField<Type>, P-33
- temporal discretisation, P-42
- Crank Nicholson, P-42
- Euler implicit, P-42
- explicit, P-42
- in OpenFOAM, P-43
- tensor, P-15
 - addition, P-17
 - algebraic operations, P-17
 - algebraic operations in OpenFOAM, P-23
 - antisymmetric, *see* tensor, skew
 - calculus, P-27
 - classes in OpenFOAM, P-23
 - cofactors, P-22
 - component average, P-20
 - component maximum, P-20
 - component minimum, P-20
 - determinant, P-22
 - deviatoric, P-21
 - diagonal, P-21
 - dimension, P-16
 - double inner product, P-19
 - geometric transformation, P-20
 - Hodge dual, P-22
 - hydrostatic, P-21
 - identities, P-21
 - identity, P-20
 - inner product, P-18
 - inverse, P-22
 - magnitude, P-20
 - magnitude squared, P-20
 - mathematics, P-15
 - notation, P-17
 - n*th power, P-20
 - outer product, P-19
 - rank, P-16
 - rank 3, P-16
 - scalar division, P-18
 - scalar multiplication, P-17
 - scale function, P-20
 - second rank, P-16
 - skew, P-21
 - square of, P-20
 - subtraction, P-17
 - symmetric, P-21
 - symmetric rank 2, P-16
 - symmetric rank 3, P-16
 - trace, P-21
 - transformation, P-20
 - transpose, P-16, P-21
 - triple inner product, P-19
 - vector cross product, P-19
- tensor class, P-23
- tensor member function
 - *, P-25
 - +, P-25
 - , P-25
 - /, P-25

- &, P-25
- &&, P-25
- ^, P-25
- cmptAv, P-25
- cofactors, P-25
- det, P-25
- dev, P-25
- diag, P-25
- I, P-25
- inv, P-25
- mag, P-25
- magSqr, P-25
- max, P-25
- min, P-25
- pow, P-25
- scale, P-25
- skew, P-25
- sqr, P-25
- symm, P-25
- T(), P-25
- tr, P-25
- transform, P-25
- tensorField class, P-29
- tensorThirdField class, P-29
- tetDecomposition utility, U-88
- tetgenToFoam utility, U-87
- text box
 - Case Name, U-122
 - Case Root, U-122
 - Lower and Upper Times, U-161
 - Opacity, U-162
 - Time step, U-161
 - times, U-31
- thermalProperties*
 - dictionary, U-50, U-51
- thermophysical
 - library, U-175
- thermophysicalFunctions
 - library, U-93
- thermophysicalProperties*
 - dictionary, U-175
- thermoType keyword, U-175
- Time window, U-29
- time
 - control, U-102
- time derivative, P-37
 - first, P-39
 - second, P-37, P-39
- Time step text box, U-161
- time step, U-25
- timeFormat keyword, U-104
- timePrecision keyword, U-104
- times text box, U-31
- timeScheme keyword, U-105
- timeStep
 - keyword entry, U-25, U-34, U-103
- tolerance
 - solver, U-113
 - solver relative, U-113
- tools
 - adjustPhi, U-91
 - algorithms, U-91
 - bound, U-91
 - compressible, U-91
 - containers, U-91
 - db, U-91
 - dimensionSet, U-91
 - dimensionedTypes, U-91
 - fields, U-91
 - finiteVolume, U-91
 - global, U-91
 - incompressible, U-91
 - interpolations, U-91
 - matrices, U-91
 - meshes, U-91
 - primitives, U-91
 - wallDist, U-91
- topoSetSource keyword, U-57
- totalPressure
 - boundary condition, U-143
- tr
 - tensor member function, P-25
- trace, *see* tensor, trace
- transform
 - tensor member function, P-25
- transportProperties*
 - dictionary, U-24, U-40, U-42
 - file, U-58
- triple inner product, P-19
- triSurface
 - library, U-92
- turbFoam solver, U-19, U-85
- turbulence
 - dissipation, U-41
 - kinetic energy, U-41
 - length scale, U-41
 - model, U-42
- turbulence keyword, U-178
- turbulence model, U-41
- turbulenceModel keyword, U-178
- turbulenceProperties*
 - dictionary, U-42, U-178
- turbulent flow
 - steady, P-54
- turbulentInlet
 - boundary condition, U-143
- tutorials
 - breaking of a dam, U-55

- lid-driven cavity flow, U-19
 - stress analysis of plate with hole, U-44
- tutorials directory, P-45, U-19
- type keyword, U-139

U

- U field, U-25
- Ucomponents utility, P-71, U-35, U-89
- UMIST
 - keyword entry, U-107
- uncompressed
 - keyword entry, U-104
- uncorrected
 - keyword entry, U-109, U-110
- uniform model, U-93
- uniform keyword, U-171
- units
 - of measurement, P-25
 - S.I. base, P-25
- uprime utility, U-89
- upwind
 - keyword entry, U-108, U-110
- upwind differencing, P-38, U-60
- Use parallel projection button, U-26, U-162
- utility
 - Co, U-88
 - FoamX, U-86
 - Lambda2, U-89
 - Mach, U-89
 - Pe, U-89
 - Q, U-89
 - Rcomponents, U-89
 - R, U-89
 - SI, U-90
 - Ucomponents, P-71, U-35, U-89
 - adiabaticFlameT, U-90
 - attachMesh, U-87
 - autoPatch, U-87
 - blockMesh, U-39, U-86, U-144
 - boxTurb, U-86
 - cellSet, U-87
 - cfxToFoam, U-87, U-151
 - checkMesh, U-87, U-152
 - checkYPlus, U-89
 - couplePatches, U-87
 - createPatch, U-87
 - decomposePar, U-80, U-81, U-90
 - deformedGeom, U-87
 - divU, U-88
 - dxFoamExec, U-88
 - engineCompRatio, U-89
 - engineSwirl, U-86
 - ensight74FoamExec, U-88, U-168
 - ensight76FoamExec, U-88
- enstrophy, U-89
- equilibriumCO, U-90
- equilibriumFlameT, U-90
- estimateScalarError, U-90
- faceSet, U-87
- fieldToCellSet, U-87
- flattenMesh, U-87
- fluentMeshToFoam, U-87, U-151
- foamConvert21To22, U-90
- foamDataToFluent, U-88, U-165
- foamDebugSwitches, U-90
- foamInfoExec, U-90
- foamMeshToFluent, U-87, U-165
- foamToDX, U-88
- foamToEnsignt, U-88
- foamToFieldview9, U-88
- foamToFieldview, U-88
- foamToVTK, U-88
- gambitToFoam, U-87, U-151
- gmshToFoam, U-87
- icoErrorEstimate, U-90
- icoMomentError, U-90
- ideasToFoam, U-87, U-151
- insideCells, U-87
- kivaToFoam, U-87
- liftDrag, U-89
- magGradU, U-89
- magU, U-35, U-89
- makePolyMesh, U-86
- mapFields, U-33, U-39, U-42, U-54, U-86, U-157
- mergeMeshes, U-87
- mirrorMesh, U-87
- mixtureAdiabaticFlameT, U-90
- momentScalarError, U-90
- moveEngineMesh, U-87
- moveMesh, U-87
- mshToFoam, U-87
- objToVTK, U-87
- pointSet, U-87
- postChannel, U-89
- ptot, U-89
- reconstructPar, U-84, U-90
- refineMesh, U-87
- refineShapeMesh, U-87
- renumberMesh, U-88
- sammToFoam, U-87
- sampleSurface, U-90
- sample, U-90, U-169
- scalePoints, U-88, U-154
- setFields, U-57, U-58
- smapToFoam, U-88
- splitMeshRegions, U-88
- splitMesh, U-88

starToFoam, [U-87](#), [U-151](#)
 stitchMesh, [U-88](#)
 streamFunction, [U-89](#)
 stressComponents, [U-89](#)
 subsetMesh, [U-88](#)
 tetDecomposition, [U-88](#)
 tetgenToFoam, [U-87](#)
 uprime, [U-89](#)
 vorticity, [U-89](#)
 wallGradU, [U-89](#)
 wallShearStress, [U-89](#)
 wdot, [U-90](#)
 writeMeshObj, [U-87](#)
 yPlusLES, [U-89](#)
 zipUpMesh, [U-88](#)

V

value keyword, [U-142](#)
 valueFraction keyword, [U-142](#)
 van Leer differencing, [P-38](#)
 vanLeer
 keyword entry, [U-108](#)
 vector, [P-16](#)
 operator, [P-27](#)
 unit, [P-20](#)
 vector class, [P-23](#), [U-101](#)
 vector product, *see* tensor, vector cross product
 vectorField class, [P-29](#)
 version keyword, [U-99](#)
 vertices keyword, [U-22](#), [U-145](#)
 veryInhomogeneousMixture model, [U-92](#), [U-176](#)
 View menu, [U-29](#), [U-162](#)
 viscosity
 kinematic, [U-24](#), [U-42](#)
 volField<Type> template class, [P-33](#)
 vorticity utility, [U-89](#)
 vtkFoam
 library, [U-159](#)

W

wall
 boundary condition, [P-64](#), [P-69](#), [U-140](#)
 wall boundary type, [U-41](#)
 wall
 keyword entry, [U-141](#)
 wall function, [U-93](#), [U-94](#)
 wallBuoyantPressure
 boundary condition, [U-143](#)
 wallDist tools, [U-91](#)
 wallFunctionCoeffs keyword, [U-178](#)
 wallGradU utility, [U-89](#)
 wallShearStress utility, [U-89](#)
 water model, [U-93](#)
 wclean script/alias, [U-73](#)

wdot utility, [U-90](#)
 wedge
 boundary condition, [U-136](#), [U-141](#), [U-150](#)
 wedge
 keyword entry, [U-141](#)
 window
 Fields, [U-29](#)
 Region, [U-29](#)
 Seed, [U-163](#)
 Selection Window, [U-26](#), [U-160](#)
 Time, [U-29](#)
 window panel
 Annotate, [U-26](#), [U-162](#)
 Camera Controls, [U-162](#)
 Camera Orientation, [U-162](#)
 Camera, [U-162](#)
 Display, [U-26](#), [U-29](#), [U-160](#), [U-161](#)
 General, [U-162](#)
 Information, [U-160](#)
 Parameters, [U-29](#), [U-160](#), [U-161](#)
 Standard Views, [U-162](#)
 Stored Camera Position, [U-162](#)
 Wireframe
 menu entry, [U-162](#)
 WM_ARCH
 environment variable, [U-74](#)
 WM_COMPILE_OPTION
 environment variable, [U-74](#)
 WM_COMPILER
 environment variable, [U-74](#)
 WM_COMPILER_BIN
 environment variable, [U-74](#)
 WM_COMPILER_DIR
 environment variable, [U-74](#)
 WM_COMPILER_LIB
 environment variable, [U-74](#)
 WM_DIR
 environment variable, [U-74](#)
 WM_JAVAC_OPTION
 environment variable, [U-74](#)
 WM_LINK_LANGUAGE
 environment variable, [U-74](#)
 WM_MPLIB
 environment variable, [U-74](#)
 WM_OPTIONS
 environment variable, [U-74](#)
 WM_PROJECT
 environment variable, [U-74](#)
 WM_PROJECT_DIR
 environment variable, [U-74](#)
 WM_PROJECT_INST_DIR
 environment variable, [U-74](#)
 WM_PROJECT_LANGUAGE
 environment variable, [U-74](#)

WM_PROJECT_USER_DIR
 environment variable, [U-74](#)
 WM_PROJECT_VERSION
 environment variable, [U-74](#)
 WM_SHELL
 environment variable, [U-74](#)
 wmake
 platforms, [U-71](#)
 wmake script/alias, [U-69](#)
 word class, [P-25](#), [P-31](#)
 writeCompression keyword, [U-104](#)
 writeControl
 keyword entry, [U-103](#)
 writeControl keyword, [U-25](#), [U-59](#), [U-103](#)
 writeFormat keyword, [U-54](#), [U-104](#)
 writeInterval keyword, [U-25](#), [U-34](#), [U-103](#)
 writeMeshObj utility, [U-87](#)
 writeNow
 keyword entry, [U-103](#)
 writePrecision keyword, [U-104](#)

X

x
 keyword entry, [U-171](#)
 XiFoam solver, [U-85](#)
 xmgr
 keyword entry, [U-104](#), [U-170](#)
 Xoodles solver, [U-85](#)
 xyz
 keyword entry, [U-171](#)

Y

y
 keyword entry, [U-171](#)
 yPlusLES utility, [U-89](#)

Z

z
 keyword entry, [U-171](#)
 zeroGradient
 boundary condition, [U-142](#)
 zipUpMesh utility, [U-88](#)
 zlib-1.2.1
 library, [U-92](#)