



The Open Source CFD Toolbox

Programmer's Guide

Version 3.0.0
2nd November 2015

Copyright © 2011-2015 OpenFOAM Foundation Ltd.
Author: Christopher J. Greenshields, CFD Direct Ltd.

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

Typeset in L^AT_EX.

License

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

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

1. Definitions

- a. “Adaptation” means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image (“synching”) will be considered an Adaptation for the purpose of this License.
- b. “Collection” means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.
- c. “Distribute” means to make available to the public the original and copies of the Work through sale or other transfer of ownership.
- d. “Licensor” means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.
- e. “Original Author” means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the

publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

- f. “Work” means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.
- g. “You” means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
- h. “Publicly Perform” means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.
- i. “Reproduce” means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights.

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

3. License Grant.

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

- a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

- b. and, to Distribute and Publicly Perform the Work including as incorporated in Collections.

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

4. Restrictions.

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

- a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested.
- b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
- c. If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution (“Attribution Parties”) in Licensor’s copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or

Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d. For the avoidance of doubt:

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

5. Representations, Warranties and Disclaimer

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

6. Limitation on Liability.

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

7. Termination

- a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collections

from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

- b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

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

Trademarks

ANSYS is a registered trademark of ANSYS Inc.

CFX is a registered trademark of Ansys Inc.

CHEMKIN is a registered trademark of Reaction Design Corporation

EnSight is a registered trademark of Computational Engineering International Ltd.

Fieldview is a registered trademark of Intelligent Light

Fluent is a registered trademark of Ansys Inc.

GAMBIT is a registered trademark of Ansys Inc.

Icem-CFD is a registered trademark of Ansys Inc.

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

JAVA is a registered trademark of Sun Microsystems Inc.

Linux is a registered trademark of Linus Torvalds

OpenFOAM is a registered trademark of ESI Group

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
1. Definitions	P-2
2. Fair Dealing Rights.	P-3
3. License Grant.	P-3
4. Restrictions.	P-4
5. Representations, Warranties and Disclaimer	P-5
6. Limitation on Liability.	P-5
7. Termination	P-5
8. Miscellaneous	P-6
Trademarks	P-7
Contents	P-9
1 Tensor mathematics	P-13
1.1 Coordinate system	P-13
1.2 Tensors	P-13
1.2.1 Tensor notation	P-15
1.3 Algebraic tensor operations	P-16
1.3.1 The inner product	P-16
1.3.2 The double inner product of two tensors	P-17
1.3.3 The triple inner product of two third rank tensors	P-17
1.3.4 The outer product	P-17
1.3.5 The cross product of two vectors	P-18
1.3.6 Other general tensor operations	P-18
1.3.7 Geometric transformation and the identity tensor	P-19
1.3.8 Useful tensor identities	P-19
1.3.9 Operations exclusive to tensors of rank 2	P-20
1.3.10 Operations exclusive to scalars	P-21
1.4 OpenFOAM tensor classes	P-21
1.4.1 Algebraic tensor operations in OpenFOAM	P-22
1.5 Dimensional units	P-24
2 Discretisation procedures	P-25
2.1 Differential operators	P-25
2.1.1 Gradient	P-25
2.1.2 Divergence	P-26
2.1.3 Curl	P-26

2.1.4	Laplacian	P-26
2.1.5	Temporal derivative	P-26
2.2	Overview of discretisation	P-27
2.2.1	OpenFOAM lists and fields	P-27
2.3	Discretisation of the solution domain	P-28
2.3.1	Defining a mesh in OpenFOAM	P-29
2.3.2	Defining a <code>geometricField</code> in OpenFOAM	P-30
2.4	Equation discretisation	P-31
2.4.1	The Laplacian term	P-36
2.4.2	The convection term	P-36
2.4.3	First time derivative	P-37
2.4.4	Second time derivative	P-37
2.4.5	Divergence	P-37
2.4.6	Gradient	P-38
2.4.7	Grad-grad squared	P-39
2.4.8	Curl	P-39
2.4.9	Source terms	P-39
2.4.10	Other explicit discretisation schemes	P-39
2.5	Temporal discretisation	P-40
2.5.1	Treatment of temporal discretisation in OpenFOAM	P-41
2.6	Boundary Conditions	P-41
2.6.1	Physical boundary conditions	P-42
3	Examples of the use of OpenFOAM	P-43
3.1	Flow around a cylinder	P-43
3.1.1	Problem specification	P-44
3.1.2	Note on <code>potentialFoam</code>	P-45
3.1.3	Mesh generation	P-45
3.1.4	Boundary conditions and initial fields	P-47
3.1.5	Running the case	P-48
3.2	Steady turbulent flow over a backward-facing step	P-50
3.2.1	Problem specification	P-50
3.2.2	Mesh generation	P-53
3.2.3	Boundary conditions and initial fields	P-56
3.2.4	Case control	P-57
3.2.5	Running the case and post-processing	P-57
3.3	Supersonic flow over a forward-facing step	P-58
3.3.1	Problem specification	P-58
3.3.2	Mesh generation	P-59
3.3.3	Running the case	P-61
3.3.4	Exercise	P-61
3.4	Decompression of a tank internally pressurised with water	P-61
3.4.1	Problem specification	P-62
3.4.2	Mesh Generation	P-63
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

Contents	P-11
3.5.1 Problem specification	P-67
3.5.2 Mesh generation	P-69
3.5.3 Running the case	P-70
Index	P-73

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.

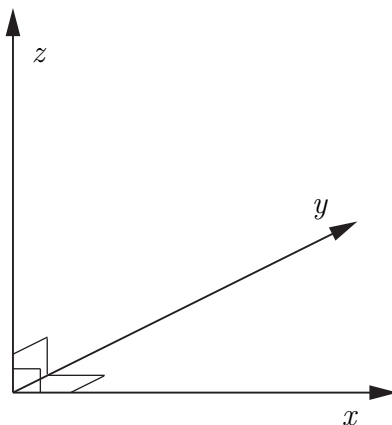


Figure 1.1: Right handed axes

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 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 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 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.

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 the 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.* **a**. This actively promotes the concept of a tensor as a 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 **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 **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

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

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 $s = \mathbf{P} \cdot \mathbf{Q}$ 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 $\mathbf{P} = \mathbf{a}\mathbf{T}$ 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 $\mathbf{a}^2 = \mathbf{a}\mathbf{a}$.

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 $\mathbf{a}^3 = \mathbf{a}\mathbf{a}\mathbf{a}$.

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} .

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

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

$$e_{ijk}e_{irs} = \delta_{jr}\delta_{ks} - \delta_{js}\delta_{kr} \quad (1.30)$$

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

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

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

$$\text{tr } \mathbf{T} = T_{11} + T_{22} + T_{33} \quad (1.32)$$

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

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

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:

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

Determinant The determinant of a second rank tensor is evaluated by

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

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

$$\begin{vmatrix} \cancel{T_{11}} & \cancel{T_{12}} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{vmatrix} = \begin{vmatrix} T_{21} & T_{23} \\ T_{31} & T_{33} \end{vmatrix} = T_{21}T_{33} - T_{23}T_{31} \quad (1.36)$$

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

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

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

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

Inverse The inverse of a tensor can be evaluated as

$$\text{inv } \mathbf{T} = \frac{\text{cof } \mathbf{T}^T}{\det \mathbf{T}} \quad (1.39)$$

Hodge dual of a tensor is a vector whose components are

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

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

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

Positive of a scalar s is

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

Limit of a scalar s by the scalar n

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

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.

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:

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

Table 1.1: Basic tensor classes in OpenFOAM

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

We can then access the component T_{13} , or T_{xz} 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		$\mathbf{a} + \mathbf{b}$	a + b
Subtraction		$\mathbf{a} - \mathbf{b}$	a - b
Scalar multiplication		$s\mathbf{a}$	s * a
Scalar division		\mathbf{a}/s	a / s
Outer product	rank $\mathbf{a}, \mathbf{b} \geq 1$	$\mathbf{a}\mathbf{b}$	a * b
Inner product	rank $\mathbf{a}, \mathbf{b} \geq 1$	$\mathbf{a} \cdot \mathbf{b}$	a & b
Double inner product	rank $\mathbf{a}, \mathbf{b} \geq 2$	$\mathbf{a} : \mathbf{b}$	a && b
Cross product	rank $\mathbf{a}, \mathbf{b} = 1$	$\mathbf{a} \times \mathbf{b}$	a ^ b
Square		\mathbf{a}^2	sqr(a)
Magnitude squared		$ \mathbf{a} ^2$	magSqr(a)
Magnitude		$ \mathbf{a} $	mag(a)
Power	$n = 0, 1, \dots, 4$	\mathbf{a}^n	pow(a,n)
Component average	$i = 1, \dots, N$	$\overline{a_i}$	cmptAv(a)
Component maximum	$i = 1, \dots, N$	$\max(a_i)$	max(a)
Component minimum	$i = 1, \dots, N$	$\min(a_i)$	min(a)
Scale		$\text{scale}(\mathbf{a}, \mathbf{b})$	scale(a,b)
Geometric transformation	transforms \mathbf{a} using tensor \mathbf{T}		transform(T,a)

Operations exclusive to tensors of rank 2

Continued on next page

Continued from previous page

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

Sign (boolean)		$\text{sgn}(s)$	<code>sign(s)</code>
Positive (boolean)		$s \geq 0$	<code>pos(s)</code>
Negative (boolean)		$s < 0$	<code>neg(s)</code>
Limit	n scalar	$\text{limit}(s, n)$	<code>limit(s,n)</code>
Square root		\sqrt{s}	<code>sqrt(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>
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>

a, 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 (m³), 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 k¹, 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)

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

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

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

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 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.

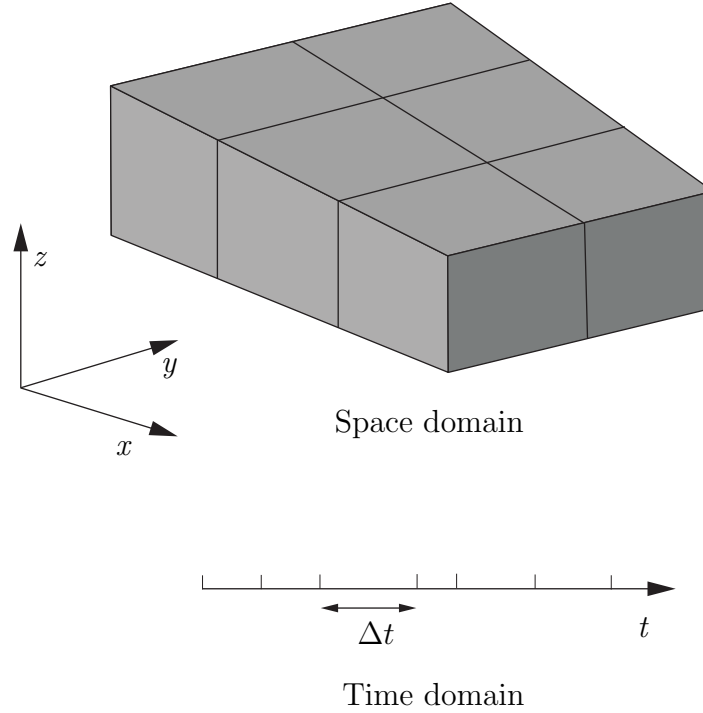


Figure 2.1: Discretisation of the solution domain

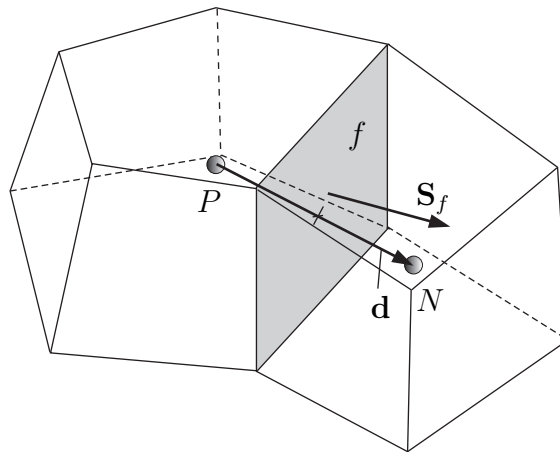


Figure 2.2: Parameters in finite volume discretisation

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 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.*.

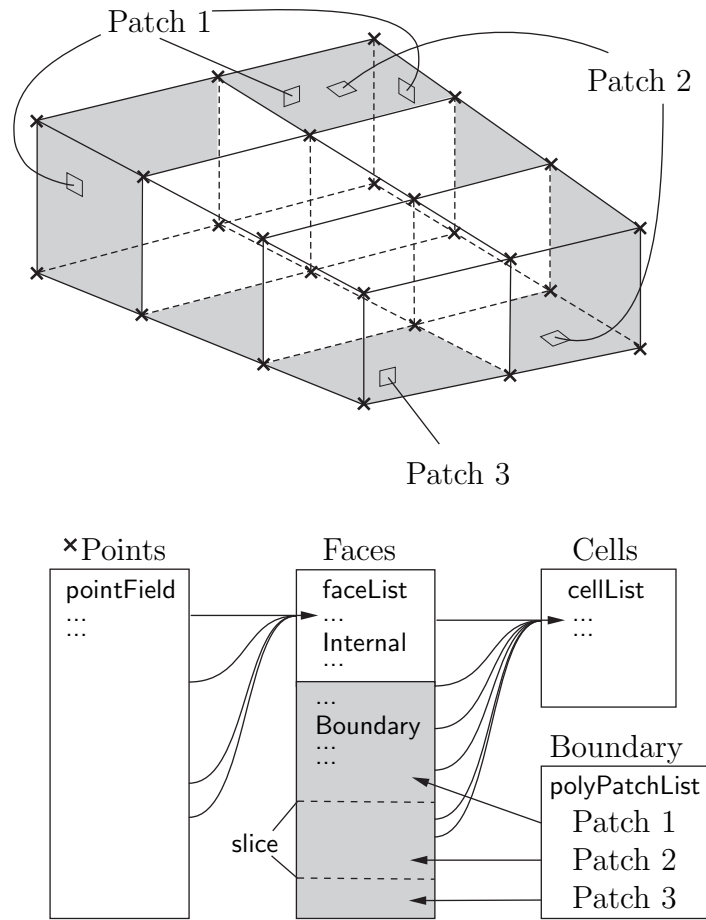


Figure 2.3: Schematic of the basic mesh description used in OpenFOAM

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 4.2.6.

Old values Discretisation of time derivatives requires field data from previous time steps.

Class	Description	Symbol	Access function
volScalarField	Cell volumes	V	<code>v()</code>
surfaceVectorField	Face area vectors	\mathbf{S}_f	<code>Sf()</code>
surfaceScalarField	Face area magnitudes	$ \mathbf{S}_f $	<code>magSf()</code>
volVectorField	Cell centres	\mathbf{C}	<code>c()</code>
surfaceVectorField	Face centres	\mathbf{C}_f	<code>Cf()</code>
surfaceScalarField	Face motion fluxes **	ϕ_g	<code>phi()</code>

Table 2.1: fvMesh stored data.

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.

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.

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`.

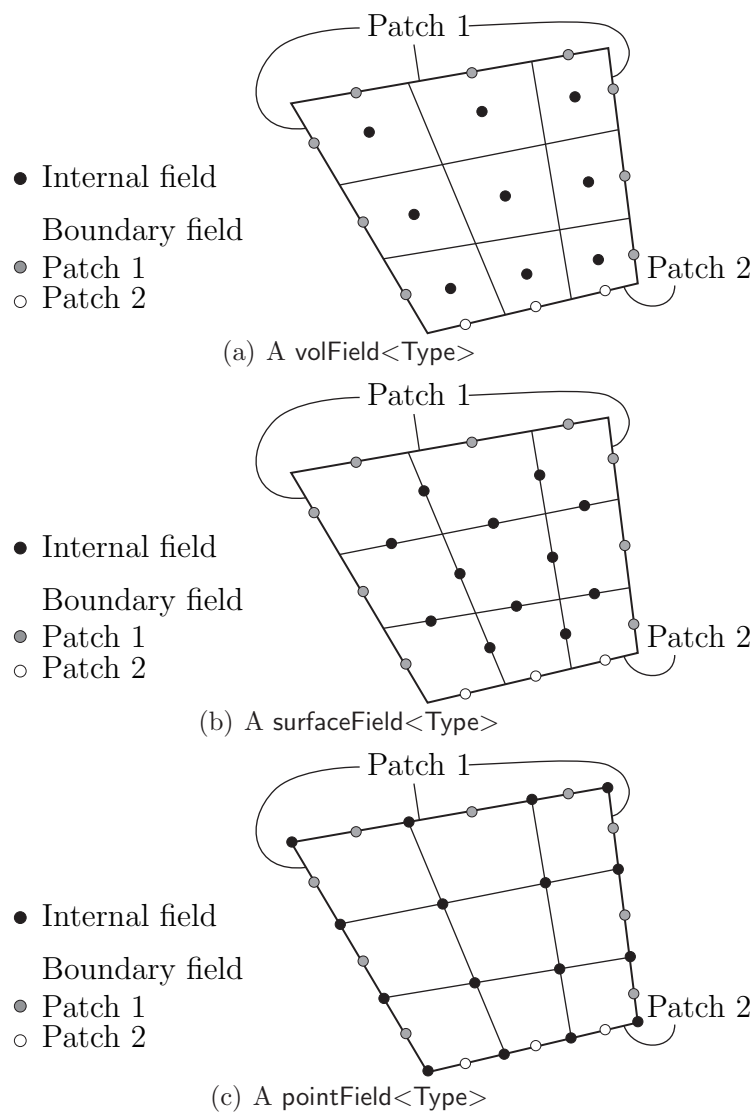


Figure 2.4: Types of `geometricField<Type>` defined on a mesh with 2 boundary patches (in 2 dimensions for simplicity)



[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` to `fvm` and `fvc` respectively. `fvm` and `fvc` 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 `fvc`, rather than one, is to distinguish:

- functions of `fvm` that calculate implicit derivatives of and return an `fvMatrix<Type>`
- some functions of `fvc` 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.

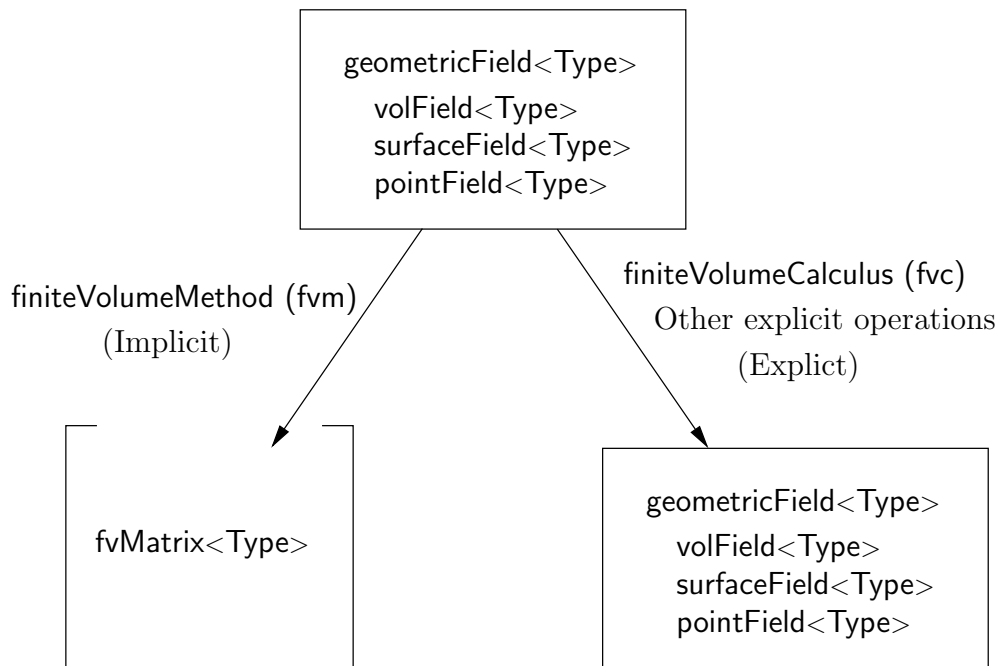


Figure 2.6: A `geometricField<Type>` and its operators

Table 2.2 lists the main functions that are available in `fvm` and `fvc` 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

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

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.

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 `fvc::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.* `fvc::gGrad`, `fvc::lsGrad` *etc.*, or by using the `fvc::grad` function combined with the appropriate `gradScheme` keyword in an input file

Gauss integration is invoked using the `fvc::grad` function with `gradScheme Gauss` or directly using the `fvc::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 `fvc::div` function, the Gaussian integration `fvc::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 `fvc::grad` function with `timeScheme leastSquares` or directly using the `fvc::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 `fvc::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 `fvc::snGradCorrection`.

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 Nicolson 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 Nicolson 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.

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.

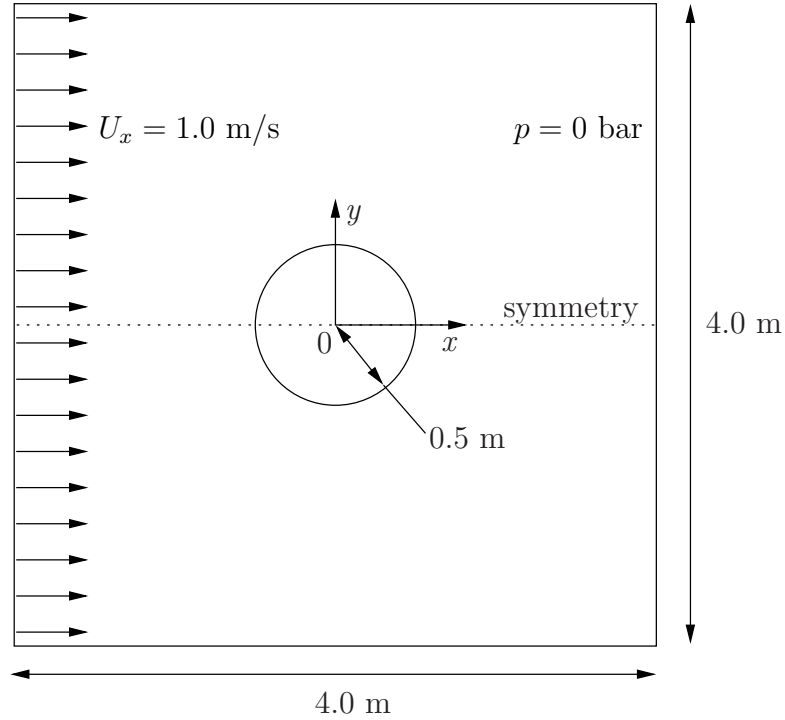


Figure 3.1: Geometry of flow round a cylinder

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 **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 meshes

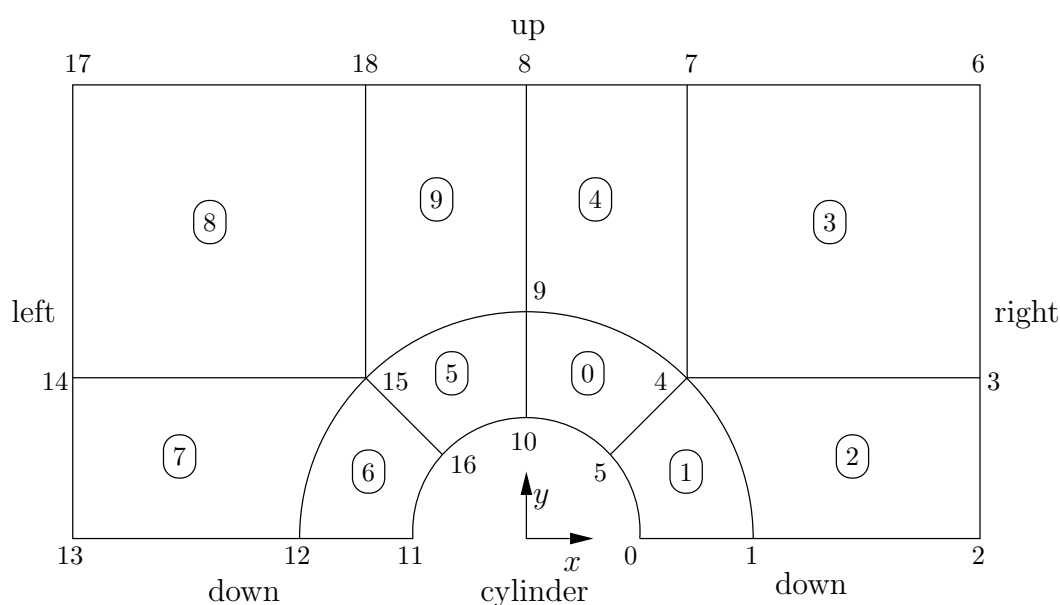


Figure 3.2: Blocks in cylinder geometry

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  /*-----* C++ *-----*/
2  |=====|
3  | \ \ \ / | F ield      | OpenFOAM: The Open Source CFD Toolbox
4  | \ \ \ / | O peration  | Version: 3.0.0
5  | \ \ \ / | A nd        | Web: www.OpenFOAM.org
6  | \ \ \ / | M anipulation|
7  /*-----*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        dictionary;

```

```

13     object      blockMeshDict;
14 }
15 // * * * * *
16
17 convertToMeters 1;
18
19 vertices #codeStream
20 {
21     codeInclude
22     #{
23         #include "pointField.H"
24     };
25
26     code
27     #{
28         pointField points(19);
29         points[0] = point(0.5, 0, -0.5);
30         points[1] = point(1, 0, -0.5);
31         points[2] = point(2, 0, -0.5);
32         points[3] = point(2, 0.707107, -0.5);
33         points[4] = point(0.707107, 0.707107, -0.5);
34         points[5] = point(0.353553, 0.353553, -0.5);
35         points[6] = point(2, 2, -0.5);
36         points[7] = point(0.707107, 2, -0.5);
37         points[8] = point(0, 2, -0.5);
38         points[9] = point(0, 1, -0.5);
39         points[10] = point(0, 0.5, -0.5);
40         points[11] = point(-0.5, 0, -0.5);
41         points[12] = point(-1, 0, -0.5);
42         points[13] = point(-2, 0, -0.5);
43         points[14] = point(-2, 0.707107, -0.5);
44         points[15] = point(-0.707107, 0.707107, -0.5);
45         points[16] = point(-0.353553, 0.353553, -0.5);
46         points[17] = point(-2, 2, -0.5);
47         points[18] = point(-0.707107, 2, -0.5);
48
49         // Duplicate z points
50         label sz = points.size();
51         points.setSize(2*sz);
52         for (label i = 0; i < sz; i++)
53         {
54             const point& pt = points[i];
55             points[i+sz] = point(pt.x(), pt.y(), -pt.z());
56         }
57
58         os << points;
59     };
60 };
61
62 blocks
63 (
64     hex (5 4 9 10 24 23 28 29) (10 10 1) simpleGrading (1 1 1)
65     hex (0 1 4 5 19 20 23 24) (10 10 1) simpleGrading (1 1 1)
66     hex (1 2 3 4 20 21 22 23) (20 10 1) simpleGrading (1 1 1)
67     hex (4 3 6 7 23 22 25 26) (20 20 1) simpleGrading (1 1 1)
68     hex (9 4 7 8 28 23 26 27) (10 20 1) simpleGrading (1 1 1)
69     hex (15 16 10 9 34 35 29 28) (10 10 1) simpleGrading (1 1 1)
70     hex (12 11 16 15 31 30 35 34) (10 10 1) simpleGrading (1 1 1)
71     hex (13 12 15 14 32 31 34 33) (20 10 1) simpleGrading (1 1 1)
72     hex (14 15 18 17 33 34 37 36) (20 20 1) simpleGrading (1 1 1)
73     hex (15 9 8 18 34 28 27 37) (10 20 1) simpleGrading (1 1 1)
74 );
75
76 edges
77 (
78     arc 0 5 (0.469846 0.17101 -0.5)
79     arc 5 10 (0.17101 0.469846 -0.5)
80     arc 1 4 (0.939693 0.34202 -0.5)
81     arc 4 9 (0.34202 0.939693 -0.5)
82     arc 19 24 (0.469846 0.17101 0.5)
83     arc 24 29 (0.17101 0.469846 0.5)
84     arc 20 23 (0.939693 0.34202 0.5)
85

```

```

86     arc 23 28 (0.34202 0.939693 0.5)
87     arc 11 16 (-0.469846 0.17101 -0.5)
88     arc 16 10 (-0.17101 0.469846 -0.5)
89     arc 12 15 (-0.939693 0.34202 -0.5)
90     arc 15 9 (-0.34202 0.939693 -0.5)
91     arc 30 35 (-0.469846 0.17101 0.5)
92     arc 35 29 (-0.17101 0.469846 0.5)
93     arc 31 34 (-0.939693 0.34202 0.5)
94     arc 34 28 (-0.34202 0.939693 0.5)
95 );
96
97 boundary
98 (
99     down
100     {
101         type symmetryPlane;
102         faces
103         (
104             (0 1 20 19)
105             (1 2 21 20)
106             (12 11 30 31)
107             (13 12 31 32)
108         );
109     }
110     right
111     {
112         type patch;
113         faces
114         (
115             (2 3 22 21)
116             (3 6 25 22)
117         );
118     }
119     up
120     {
121         type symmetryPlane;
122         faces
123         (
124             (7 8 27 26)
125             (6 7 26 25)
126             (8 18 37 27)
127             (18 17 36 37)
128         );
129     }
130     left
131     {
132         type patch;
133         faces
134         (
135             (14 13 32 33)
136             (17 14 33 36)
137         );
138     }
139     cylinder
140     {
141         type symmetry;
142         faces
143         (
144             (10 5 24 29)
145             (5 0 19 24)
146             (16 10 29 35)
147             (11 16 35 30)
148         );
149     }
150 );
151
152 mergePatchPairs
153 (
154 );
155
156 // *****

```

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  /*-----* C++ *-----*/
2  |=====|
3  | \ \ \ / | F i e l d | OpenFOAM: The Open Source CFD Toolbox
4  | \ \ \ / | O p e r a t i o n | Version: 3.0.0
5  | \ \ \ / | A n d | Web: www.OpenFOAM.org
6  | \ \ \ / | M a n i p u l a t i o n |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        controlDict;
15 }
16 // * * * * *
17
18 application      potentialFoam;
19
20 startFrom        startTime;
21
22 startTime        0;
23
24 stopAt           endTime;
25
26 endTime          1;
27
28 deltaT           1;
29
30 writeControl      timeStep;
31
32 writeInterval     1;
33
34 purgeWrite        0;
35
36 writeFormat       ascii;
37
38 writePrecision    6;
39
40 writeCompression  off;
41
42 timeFormat        general;
43
44 timePrecision     6;
45
46 runtimeModifiable true;
47
48 functions
49 {
50     difference
51     {
52         // Load the library containing the 'coded' functionObject
53         functionObjectLibs ("libutilityFunctionObjects.so");
54         type coded;
55         // Name of on-the-fly generated functionObject
56         redirectType error;
57         code

```



```

58     #{
59         // Lookup U
60         Info<< "Looking up field U\n" << endl;
61         const volVectorField& U = mesh().lookupObject<volVectorField>("U");
62
63         Info<< "Reading inlet velocity  uInfX\n" << endl;
64
65         scalar ULeft = 0.0;
66         label leftI = mesh().boundaryMesh().findPatchID("left");
67         const fvPatchVectorField& fvp = U.boundaryField()[leftI];
68         if (fvp.size())
69         {
70             ULeft = fvp[0].x();
71         }
72         reduce(ULeft, maxOp<scalar>());
73
74         dimensionedScalar uInfX
75         (
76             "uInfX",
77             dimensionSet(0, 1, -1, 0, 0),
78             ULeft
79         );
80
81         Info << "U at inlet = " << uInfX.value() << " m/s" << endl;
82
83
84         scalar magCylinder = 0.0;
85         label cylI = mesh().boundaryMesh().findPatchID("cylinder");
86         const fvPatchVectorField& cylFvp = mesh().C().boundaryField()[cylI];
87         if (cylFvp.size())
88         {
89             magCylinder = mag(cylFvp[0]);
90         }
91         reduce(magCylinder, maxOp<scalar>());
92
93         dimensionedScalar radius
94         (
95             "radius",
96             dimensionSet(0, 1, 0, 0, 0),
97             magCylinder
98         );
99
100        Info << "Cylinder radius = " << radius.value() << " m" << endl;
101
102        volVectorField UA
103        (
104            IOobject
105            (
106                "UA",
107                mesh().time().timeName(),
108                U.mesh(),
109                IOobject::NO_READ,
110                IOobject::AUTO_WRITE
111            ),
112            U
113        );
114
115        Info<< "\nEvaluating analytical solution" << endl;
116
117        const volVectorField& centres = UA.mesh().C();
118        volScalarField magCentres(mag(centres));
119        volScalarField theta(acos((centres & vector(1,0,0))/magCentres));
120
121        volVectorField cs2theta
122        (
123            cos(2*theta)*vector(1,0,0)
124            + sin(2*theta)*vector(0,1,0)
125        );
126
127        UA = uInfX*(dimensionedVector(vector(1,0,0))
128            - pow((radius/magCentres),2)*cs2theta);
129
130        // Force writing of UA (since time has not changed)
131        UA.write();
132
133        volScalarField error("error", mag(U-UA)/mag(UA));
134
135        Info<<"Writing relative error in U to " << error.objectPath()

```

```

136             << endl;
137
138             error.write();
139         #};
140     }
141 }
142
143
144 // *****

```

`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).

3.2 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.

Governing equations

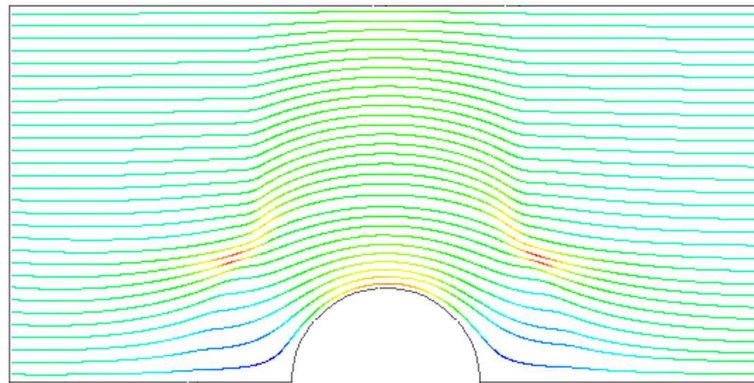
- Mass continuity for incompressible flow

$$\nabla \cdot \mathbf{U} = 0 \quad (3.3)$$

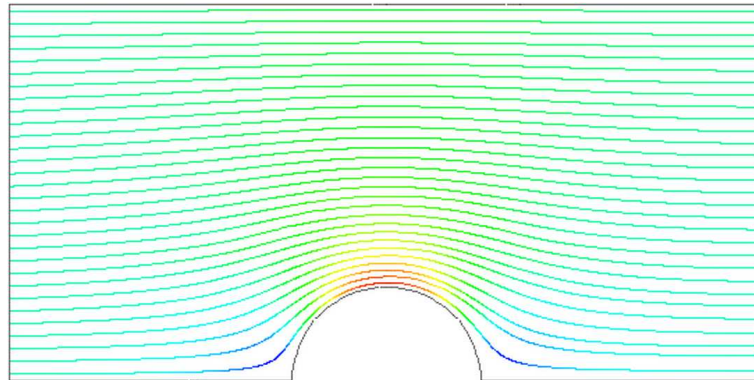
- Steady flow momentum equation

$$\nabla \cdot (\mathbf{UU}) + \nabla \cdot \mathbf{R} = -\nabla p \quad (3.4)$$

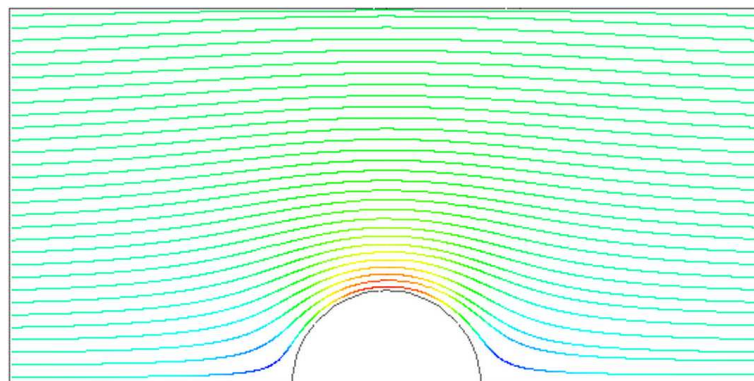
where p is kinematic pressure and (in slightly over-simplistic terms) $\mathbf{R} = \nu_{eff} \nabla \mathbf{U}$ is the viscous stress term with an effective kinematic viscosity ν_{eff} , calculated from selected transport and turbulence models.



(a) With no non-orthogonal correction



(b) With non-orthogonal correction



(c) Analytical solution

Figure 3.3: Streamlines of potential flow

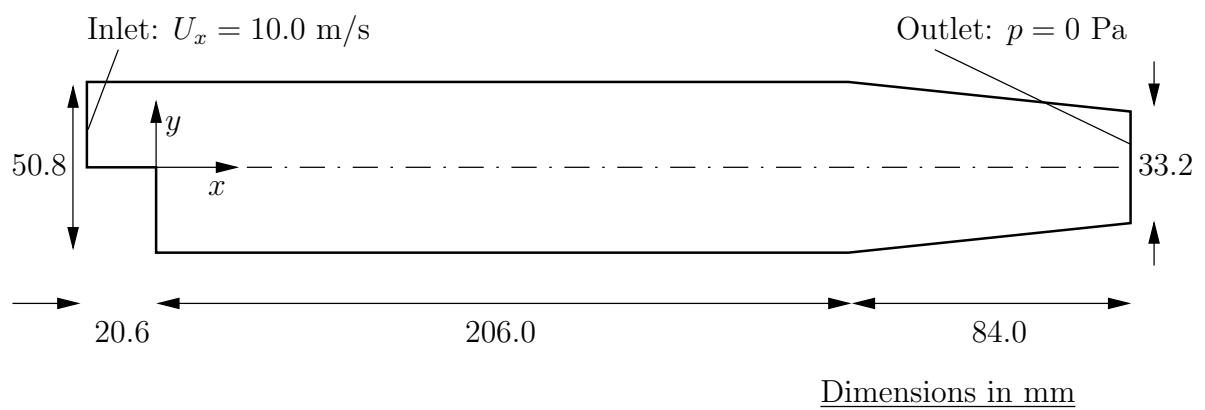


Figure 3.4: Geometry of backward-facing step

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 `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.

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  /*-----*-- C++ -*-----*/
2  |=====|
3  | \\\ / | F ield | OpenFOAM: The Open Source CFD Toolbox |
4  |  \\  | O peration | Version: 3.0.0 |
5  |   \\  | A nd | Web: www.OpenFOAM.org |
6  |    \\  | M anipulation | |
7  /*-----*--*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        blockMeshDict;

```

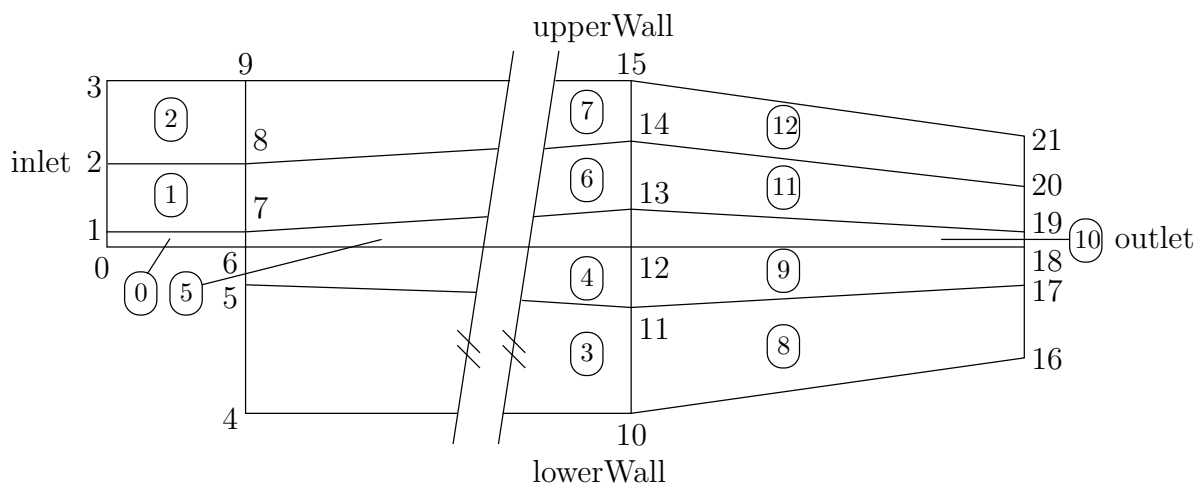


Figure 3.5: Blocks in backward-facing step

```

14 }
15 // * * * * *
16
17 convertToMeters 0.001;
18
19 vertices
20 (
21     (-20.6 0 -0.5)
22     (-20.6 3 -0.5)
23     (-20.6 12.7 -0.5)
24     (-20.6 25.4 -0.5)
25     (0 -25.4 -0.5)
26     (0 -5 -0.5)
27     (0 0 -0.5)
28     (0 3 -0.5)
29     (0 12.7 -0.5)
30     (0 25.4 -0.5)
31     (206 -25.4 -0.5)
32     (206 -8.5 -0.5)
33     (206 0 -0.5)
34     (206 6.5 -0.5)
35     (206 17 -0.5)
36     (206 25.4 -0.5)
37     (290 -16.6 -0.5)
38     (290 -6.3 -0.5)
39     (290 0 -0.5)
40     (290 4.5 -0.5)
41     (290 11 -0.5)
42     (290 16.6 -0.5)
43     (-20.6 0 0.5)
44     (-20.6 3 0.5)
45     (-20.6 12.7 0.5)
46     (-20.6 25.4 0.5)
47     (0 -25.4 0.5)
48     (0 -5 0.5)
49     (0 0 0.5)
50     (0 3 0.5)
51     (0 12.7 0.5)
52     (0 25.4 0.5)
53     (206 -25.4 0.5)
54     (206 -8.5 0.5)
55     (206 0 0.5)
56     (206 6.5 0.5)
57     (206 17 0.5)
58     (206 25.4 0.5)
59     (290 -16.6 0.5)
60     (290 -6.3 0.5)
61     (290 0 0.5)
62     (290 4.5 0.5)
63     (290 11 0.5)
64     (290 16.6 0.5)
65 );
66
67 blocks
68 (

```

```

69     hex (0 6 7 1 22 28 29 23) (18 7 1) simpleGrading (0.5 1.8 1)
70     hex (1 7 8 2 23 29 30 24) (18 10 1) simpleGrading (0.5 4 1)
71     hex (2 8 9 3 24 30 31 25) (18 13 1) simpleGrading (0.5 0.25 1)
72     hex (4 10 11 5 26 32 33 27) (180 18 1) simpleGrading (4 1 1)
73     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)
74     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)
75     hex (7 13 14 8 29 35 36 30) (180 10 1) edgeGrading (4 4 4 4 4 1 1 4 1 1 1 1)
76     hex (8 14 15 9 30 36 37 31) (180 13 1) simpleGrading (4 0.25 1)
77     hex (10 16 17 11 32 38 39 33) (25 18 1) simpleGrading (2.5 1 1)
78     hex (11 17 18 12 33 39 40 34) (25 9 1) simpleGrading (2.5 1 1)
79     hex (12 18 19 13 34 40 41 35) (25 7 1) simpleGrading (2.5 1 1)
80     hex (13 19 20 14 35 41 42 36) (25 10 1) simpleGrading (2.5 1 1)
81     hex (14 20 21 15 36 42 43 37) (25 13 1) simpleGrading (2.5 0.25 1)
82 );
83
84 edges
85 (
86 );
87
88 boundary
89 (
90     inlet
91     {
92         type patch;
93         faces
94         (
95             (0 22 23 1)
96             (1 23 24 2)
97             (2 24 25 3)
98         );
99     }
100     outlet
101     {
102         type patch;
103         faces
104         (
105             (16 17 39 38)
106             (17 18 40 39)
107             (18 19 41 40)
108             (19 20 42 41)
109             (20 21 43 42)
110         );
111     }
112     upperWall
113     {
114         type wall;
115         faces
116         (
117             (3 25 31 9)
118             (9 31 37 15)
119             (15 37 43 21)
120         );
121     }
122     lowerWall
123     {
124         type wall;
125         faces
126         (
127             (0 6 28 22)
128             (6 5 27 28)
129             (5 4 26 27)
130             (4 10 32 26)
131             (10 16 38 32)
132         );
133     }
134     frontAndBack
135     {
136         type empty;
137         faces
138         (
139             (22 28 29 23)
140             (23 29 30 24)
141             (24 30 31 25)
142             (26 32 33 27)
143             (27 33 34 28)
144             (28 34 35 29)
145             (29 35 36 30)
146             (30 36 37 31)

```

```

147         (32 38 39 33)
148         (33 39 40 34)
149         (34 40 41 35)
150         (35 41 42 36)
151         (36 42 43 37)
152         (0 1 7 6)
153         (1 2 8 7)
154         (2 3 9 8)
155         (4 5 11 10)
156         (5 6 12 11)
157         (6 7 13 12)
158         (7 8 14 13)
159         (8 9 15 14)
160         (10 11 17 16)
161         (11 12 18 17)
162         (12 13 19 18)
163         (13 14 20 19)
164         (14 15 21 20)
165     );
166 }
167 );
168
169 mergePatchPairs
170 (
171 );
172
173 // *****

```

A major feature of this problem is the use of the full mesh grading capability of **blockMesh** that is described in section 5.3.1 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 5.3.1 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

$$U'_x = U'_y = U'_z = \frac{5}{100}10 = 0.5 \text{ m/s} \quad (3.5)$$

and

$$k = \frac{3}{2}(0.5)^2 = 0.375 \text{ m}^2/\text{s}^2 \quad (3.6)$$

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

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{l} = \frac{0.09^{0.75} 0.375^{1.5}}{0.1 \times 25.4 \times 10^{-3}} = 14.855 \text{ m}^2/\text{s}^3 \quad (3.7)$$

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.

3.2.5 Running the case and post-processing

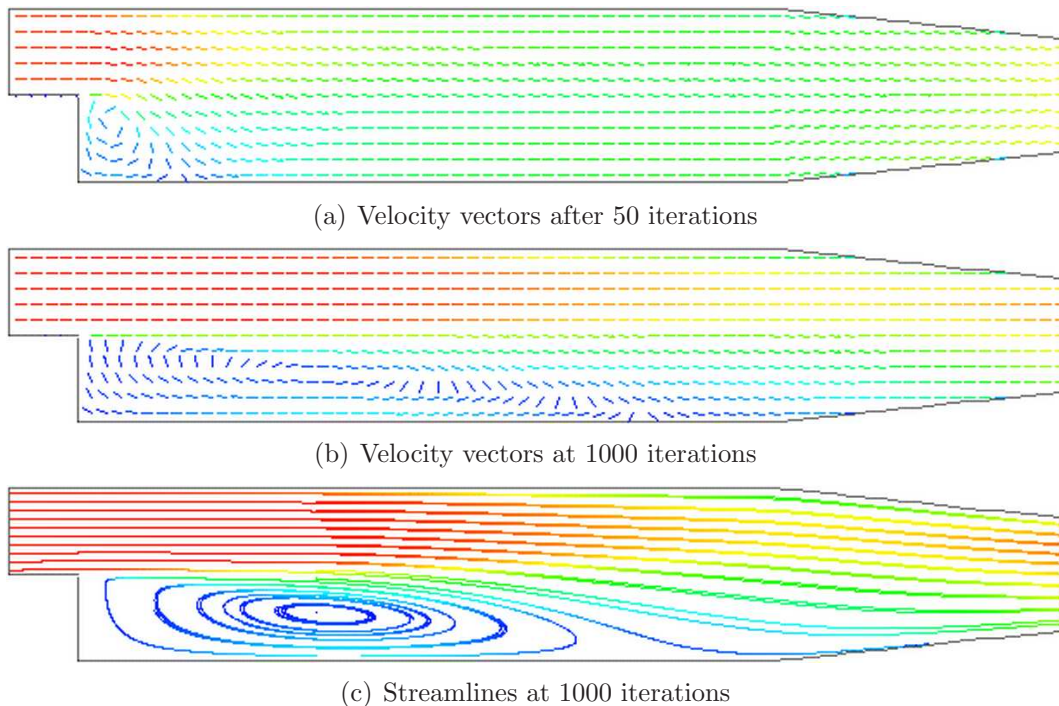


Figure 3.6: Development of a vortex in the backward-facing step.

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

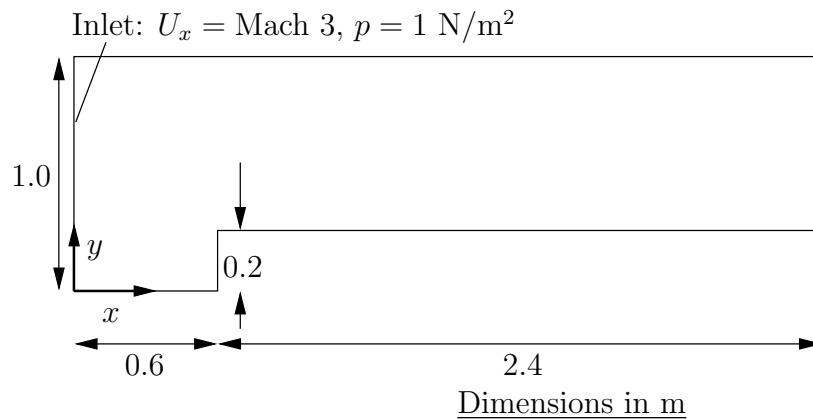


Figure 3.7: Geometry of the forward step geometry

Governing equations

- Mass continuity

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \quad (3.8)$$

- Ideal gas

$$p = \rho RT \quad (3.9)$$

- Momentum equation for Newtonian fluid

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot (\rho \mathbf{U} \mathbf{U}) - \nabla \cdot \mu \nabla \mathbf{U} = -\nabla p \quad (3.10)$$

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

$$\frac{\partial \rho e}{\partial t} + \nabla \cdot (\rho \mathbf{U} e) - \nabla \cdot \left(\frac{k}{C_v} \right) \nabla e = p \nabla \cdot \mathbf{U} \quad (3.11)$$

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.

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

$$\gamma = C_p / C_v = \frac{R}{C_v} + 1 \quad (3.12)$$

3.3.2 Mesh generation

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

```

1  /*-----*-- C++ --*-----*/
2  |=====|
3  | \\\ / | F i e l d | OpenFOAM: The Open Source CFD Toolbox |
4  | \\\ / | O p e r a t i o n | Version: 3.0.0 |
5  | \\\ / | A n d | Web: www.OpenFOAM.org |
6  | \\\ / | M a n i p u l a t i o n | |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        blockMeshDict;
14 }
15 // *****
16
17 convertToMeters 1;
18
19 vertices
20 (
21     (0 0 -0.05)
22     (0.6 0 -0.05)
23     (0 0.2 -0.05)
24     (0.6 0.2 -0.05)
25     (3 0.2 -0.05)
26     (0 1 -0.05)
27     (0.6 1 -0.05)
28     (3 1 -0.05)
29     (0 0 0.05)
30     (0.6 0 0.05)
31     (0 0.2 0.05)
32     (0.6 0.2 0.05)
33     (3 0.2 0.05)
34     (0 1 0.05)
35     (0.6 1 0.05)
36     (3 1 0.05)
37 );
38
39 blocks
40 (
41     hex (0 1 3 2 8 9 11 10) (25 10 1) simpleGrading (1 1 1)
42     hex (2 3 6 5 10 11 14 13) (25 40 1) simpleGrading (1 1 1)
43     hex (3 4 7 6 11 12 15 14) (100 40 1) simpleGrading (1 1 1)
44 );
45
46 edges
47 (
48 );
49
50 boundary
51 (
52     inlet
53     {
54         type patch;
55         faces
56         (
57             (0 8 10 2)
58             (2 10 13 5)
59         );
60     }
61     outlet
62     {
63         type patch;
64         faces
65         (
66             (4 7 15 12)
67         );
68     }
69     bottom
70     {
71         type symmetryPlane;
72         faces
73         (
74             (0 1 9 8)
75         );
76     }
77     top
78     {
79         type symmetryPlane;

```

```

80         faces
81         (
82             (5 13 14 6)
83             (6 14 15 7)
84         );
85     }
86     obstacle
87     {
88         type patch;
89         faces
90         (
91             (1 3 11 9)
92             (3 4 12 11)
93         );
94     }
95 );
96
97 mergePatchPairs
98 (
99 );
100
101 // ***** //

```

3.3.3 Running the case

The case approaches a steady-state at some time after 5 s. The results for pressure at 10 s are shown in Figure 3.8. The results clearly show discontinuities in pressure, *i.e.* shock waves, emanating from ahead of the base of the step.

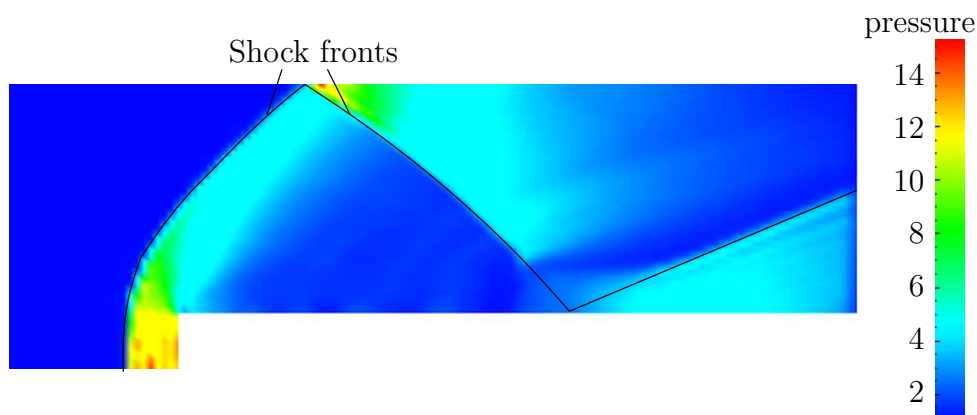


Figure 3.8: Shock fronts in the forward step problem

3.3.4 Exercise

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

3.4 Decompression of a tank internally pressurised with water

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

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

- Mesh refinement

- Pressure waves in liquids

3.4.1 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

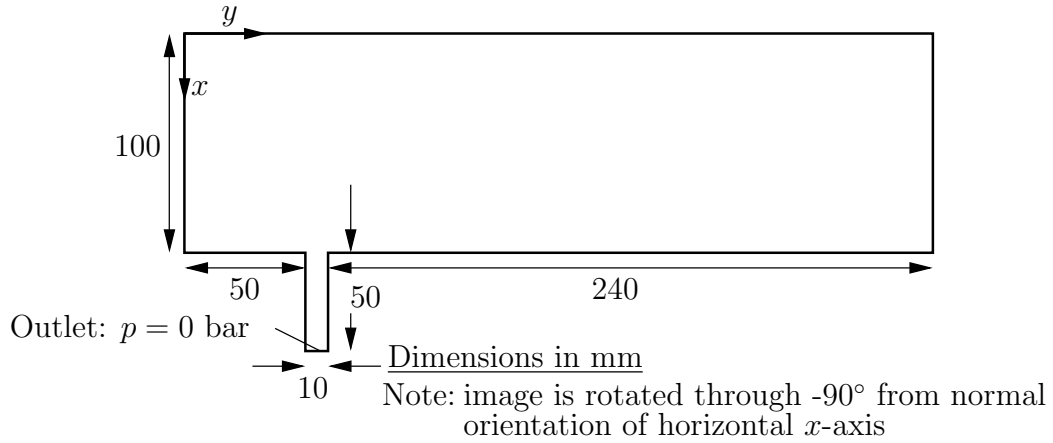


Figure 3.9: Geometry of a tank with outflow pipe

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

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \quad (3.13)$$

- The barotropic relationship

$$\frac{\partial \rho}{\partial p} = \frac{\rho}{K} = \psi \quad (3.14)$$

where K is the bulk modulus

- Equation 3.14 is linearised as

$$\rho \approx \rho_0 + \psi (p - p_0) \quad (3.15)$$

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

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot (\rho \mathbf{U} \mathbf{U}) - \nabla \cdot \mu \nabla \mathbf{U} = -\nabla p \quad (3.16)$$

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

Transport properties

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

Thermodynamic properties

- Density of water $\rho = 1000$ kg/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  /*-----*----- C++ *-----*\
2  |=====|
3  |  \ \  /  | F ield          | OpenFOAM: The Open Source CFD Toolbox
4  |  \ \  /  | O peration       | Version: 3.0.0
5  |  \ \  /  | A nd              | Web:      www.OpenFOAM.org
6  |  \ \  /  | M anipulation    |
7  \*-----*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class          dictionary;
13     object         blockMeshDict;
14 }
15 // * * * * *
16
17 convertToMeters 0.1;
18
19 vertices
20 (
21     (0 0 -0.1)
22     (1 0 -0.1)
23     (0 0.5 -0.1)
24     (1 0.5 -0.1)
25     (1.5 0.5 -0.1)
26     (0 0.6 -0.1)
27     (1 0.6 -0.1)
28     (1.5 0.6 -0.1)
29     (0 3 -0.1)
30     (1 3 -0.1)
31     (0 0 0.1)
32     (1 0 0.1)
33     (0 0.5 0.1)
34     (1 0.5 0.1)
35     (1.5 0.5 0.1)
36     (0 0.6 0.1)

```

```

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

```


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:

$$c = \sqrt{\frac{1}{\psi}} = \sqrt{\frac{1}{4.54 \times 10^{-7}}} = 1483.2 \text{ m/s.} \quad (3.17)$$

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

$$Co = \frac{U \Delta t}{\Delta x} \quad (3.18)$$

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

```

1  /*-----* C++ *-----*/
2  |=====|
3  | \ \ / \ | F ield      | OpenFOAM: The Open Source CFD Toolbox
4  | \ \ / \ | O peration  | Version: 3.0.0
5  | \ \ / \ | A nd        | Web:      www.OpenFOAM.org
6  | \ \ / \ | M anipulation|
7  \*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        controlDict;
15 }
16 // * * * * *
17
18 application      sonicLiquidFoam;
19
20 startFrom        startTime;
21
22 startTime        0;
23
24 stopAt           endTime;
25
26 endTime          0.0001;
27
28 deltaT           5e-07;
29
30 writeControl      timeStep;
31
32 writeInterval     20;
33
34 purgeWrite        0;
35
36 writeFormat       ascii;
37
38 writePrecision    6;
39

```

```

40 writeCompression off;
41 timeFormat      general;
42 timePrecision   6;
43 runTimeModifiable true;
44
45 // *****
46

```

3.4.4 Running the case

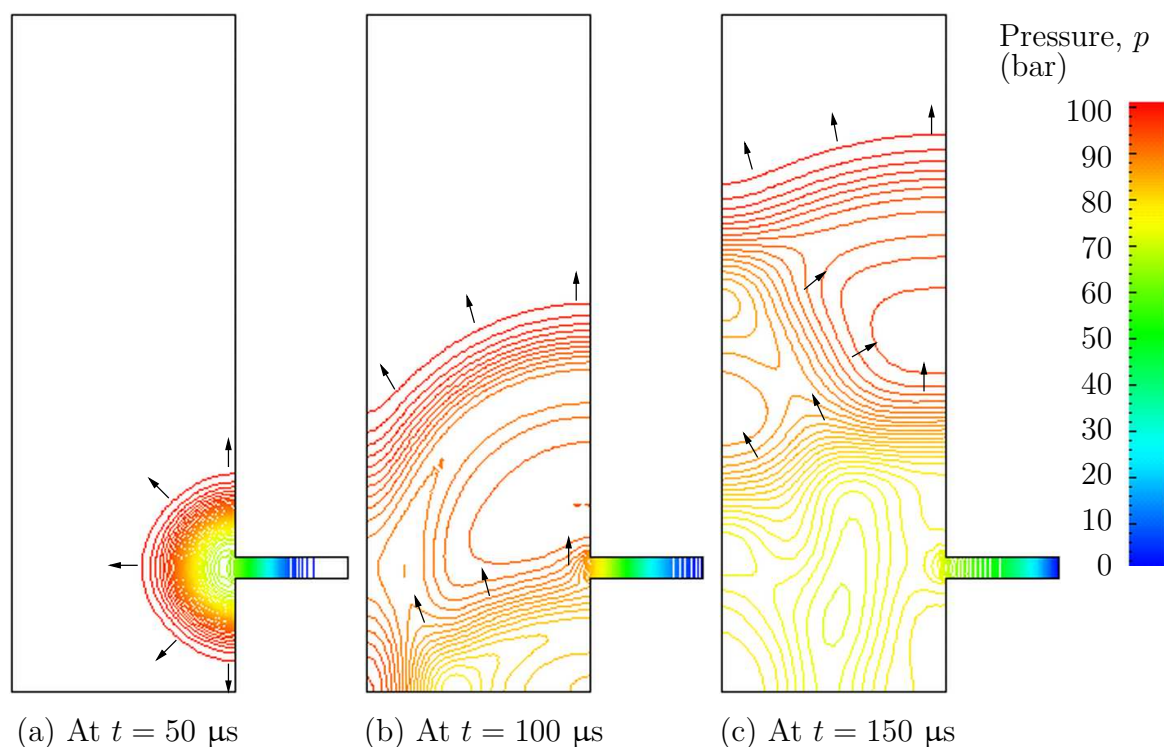


Figure 3.10: Propagation of pressure waves

The user can run the case and view results in **dxFoam**. The liquid flows out through the nozzle causing a wave to move along the nozzle. As it reaches the inlet to the tank, some of the wave is transmitted into the tank and some of it is reflected. While a wave is reflected up and down the inlet pipe, the waves transmitted into the tank expand and propagate through the tank. In Figure 3.10, 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.

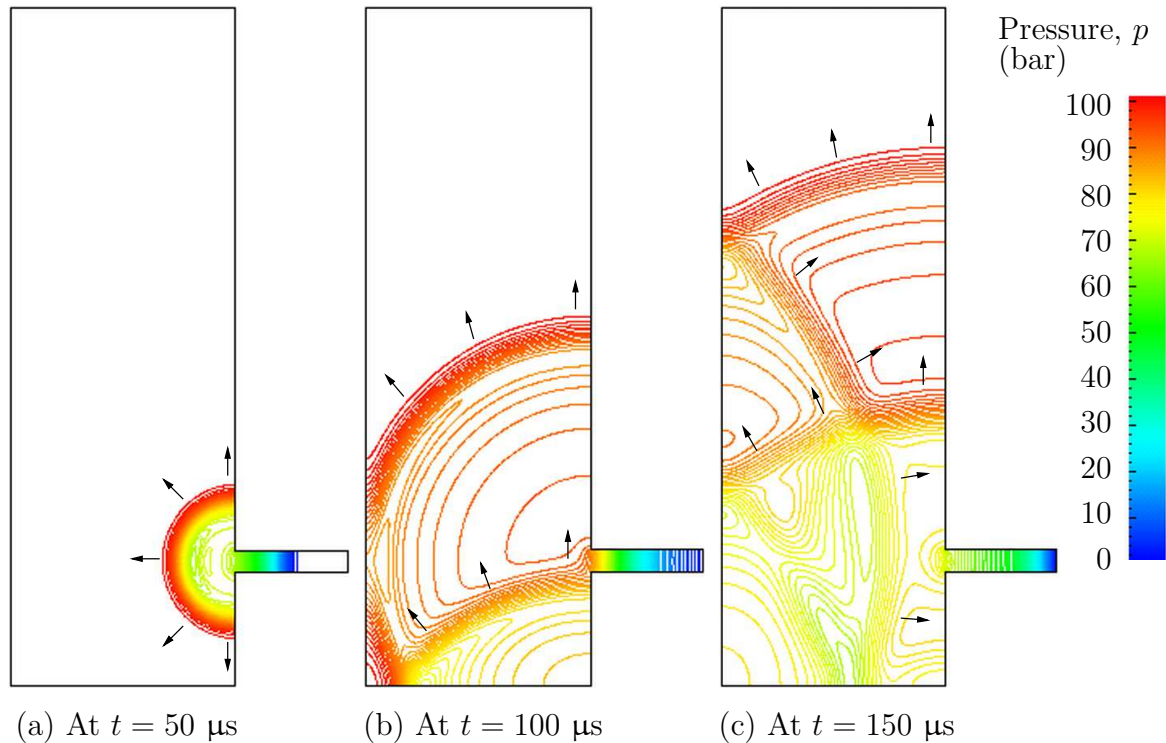


Figure 3.11: Propagation of pressure waves with refined mesh

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`.

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.

Governing equations

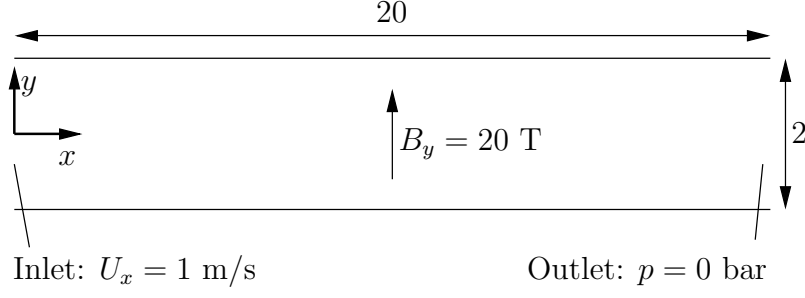


Figure 3.12: Geometry of the Hartmann problem

- Mass continuity for incompressible fluid

$$\nabla \cdot \mathbf{U} = 0 \quad (3.19)$$

- Momentum equation for incompressible fluid

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) + \nabla \cdot (2\mathbf{B}\Gamma_{\mathbf{B}\mathbf{U}}\mathbf{B}) + \nabla \cdot (\nu\mathbf{U}) + \nabla (\Gamma_{\mathbf{B}\mathbf{U}}\mathbf{B} : \mathbf{B}) = -\nabla p \quad (3.20)$$

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

- Maxwell's equations

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \quad (3.21)$$

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

$$\nabla \cdot \mathbf{B} = 0 \quad (3.22)$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t} = \mathbf{J} \quad (3.23)$$

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

- Charge continuity

$$\nabla \cdot \mathbf{J} = 0 \quad (3.24)$$

- Constitutive law

$$\mathbf{B} = \mu \mathbf{H} \quad (3.25)$$

- Ohm's law

$$\mathbf{J} = \sigma (\mathbf{E} + \mathbf{U} \times \mathbf{B}) \quad (3.26)$$

- Combining Equation 3.21, Equation 3.23, Equation 3.26, and taking the curl

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{B}) - \nabla \cdot (\phi_{\mathbf{B}}\mathbf{U}) - \nabla \cdot (\Gamma_{\mathbf{B}}\mathbf{B}) = 0 \quad (3.27)$$

Boundary conditions

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

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

Transport properties

- Kinematic viscosity $\nu = 1$ Pa s
- Density $\rho = 1$ kg m/s
- Electrical conductivity $\sigma = 1$ (Ω m) $^{-1}$
- Permeability $\mu = 1$ H/m

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

Case name `hartmann` case located in the `$FOAM_TUTORIALS/mhdFoam` directory.

3.5.2 Mesh generation

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

```

1  /*----- C++ -----*/
2  |=====|
3  | \ \ \ / | F i e l d | OpenFOAM: The Open Source CFD Toolbox
4  | \ \ \ / | O p e r a t i o n | Version: 3.0.0
5  | \ \ \ / | A n d | Web: www.OpenFOAM.org
6  | \ \ \ / | M a n i p u l a t i o n |
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        blockMeshDict;
14 }
15 // *****
16
17 convertToMeters 1;
18
19 vertices
20 (
21     (0 -1 0)
22     (20 -1 0)
23     (20 1 0)
24     (0 1 0)
25     (0 -1 0.1)
26     (20 -1 0.1)
27     (20 1 0.1)
28     (0 1 0.1)
29 );
30
31 blocks
32 (
33     hex (0 1 2 3 4 5 6 7) (100 40 1) simpleGrading (1 1 1)
34 );
35
36 edges

```

```

37  (
38  );
39
40  boundary
41  (
42      inlet
43      {
44          type patch;
45          faces
46          (
47              (0 4 7 3)
48          );
49      }
50      outlet
51      {
52          type patch;
53          faces
54          (
55              (2 6 5 1)
56          );
57      }
58      lowerWall
59      {
60          type patch;
61          faces
62          (
63              (1 5 4 0)
64          );
65      }
66      upperWall
67      {
68          type patch;
69          faces
70          (
71              (3 7 6 2)
72          );
73      }
74      frontAndBack
75      {
76          type empty;
77          faces
78          (
79              (0 3 2 1)
80              (4 5 6 7)
81          );
82      }
83  );
84
85  mergePatchPairs
86  (
87  );
88
89  // *****

```

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 **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

$$M = BL\sqrt{\frac{\sigma}{\rho\nu}} \quad (3.28)$$

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 **B** to 1 T and re-run the code and **Ucomponents**. In this case, $M = 1$ and the electromagnetic body forces no longer dominate.

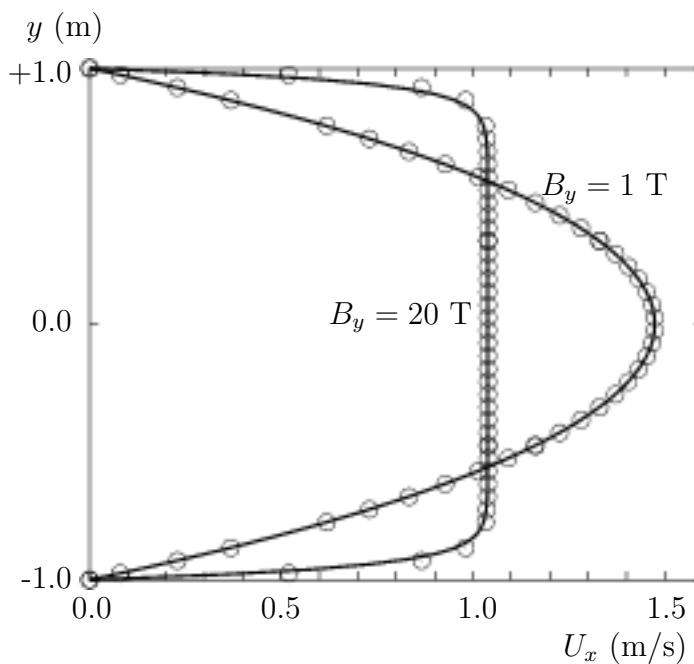


Figure 3.13: Velocity profile in the Hartmann problem for $B_y = 1$ T and $B_y = 20$ T.

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:

$$\frac{U_x(y)}{U_x(0)} = \frac{\cosh M - \cosh M(y/L)}{\cosh M - 1} \quad (3.29)$$

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-23
 - `+`
 - tensor member function, P-23
 - `-`
 - tensor member function, P-23
 - `/`
 - tensor member function, P-23
 - `/*...*/`
 - C++ syntax, U-78
 - `//`
 - C++ syntax, U-78
 - OpenFOAM file syntax, U-108
 - `# include`
 - C++ syntax, U-72, U-78
 - `&`
 - tensor member function, P-23
 - `&&`
 - tensor member function, P-23
 - `^`
 - tensor member function, P-23
 - `<LESModel>Coeffs` keyword, U-200
 - `<RASModel>Coeffs` keyword, U-200
 - `<delta>Coeffs` keyword, U-200
 - `0.000000e+00` directory, U-108
 - 1-dimensional mesh, U-135
 - 1D mesh, U-135
 - 2-dimensional mesh, U-135
 - 2D mesh, U-135
- ## Numbers
- `0` directory, U-108
- ## A
- access functions, P-21
 - `addLayersControls` keyword, U-152
 - `adiabaticFlameT` utility, U-98
 - `adiabaticPerfectFluid` model, U-102, U-197
 - `adjointShapeOptimizationFoam` solver, U-86
 - `adjustableRunTime`
 - keyword entry, U-62, U-116
 - `adjustTimeStep` keyword, U-62, U-117
 - `agglomerator` keyword, U-127
 - algorithms tools, U-99
 - `alphaContactAngle`
 - boundary condition, U-59
 - analytical solution, P-43
 - Animations window panel, U-176
 - `anisotropicFilter` model, U-104
 - Annotation window panel, U-24
 - `ansysToFoam` utility, U-92
 - `APIfunctions` model, U-103
 - applications, U-69
 - Apply button, U-172, U-176
 - `applyBoundaryLayer` utility, U-91
 - `applyWallFunctionBoundaryConditions` utility, U-91
 - arbitrarily unstructured, P-29
 - `arc`
 - keyword entry, U-144
 - `arc` keyword, U-143
 - `As` keyword, U-195
 - `ascii`
 - keyword entry, U-116
 - `attachMesh` utility, U-93
 - Auto Accept button, U-176
 - `autoMesh`
 - library, U-100
 - `autoPatch` utility, U-93
 - `autoRefineMesh` utility, U-94
 - axes
 - right-handed, U-142
 - right-handed rectangular Cartesian, P-13, U-18
 - axi-symmetric cases, U-139, U-150
 - axi-symmetric mesh, U-135

B

- background
 - process, U-24, U-81
- backward
 - keyword entry, U-124
- Backward differencing, P-37
- barotropicCompressibilityModels
 - library, U-102
- basicMultiComponentMixture model, U-101
- basicSolidThermo
 - library, U-103
- basicThermophysicalModels
 - library, U-101
- binary
 - keyword entry, U-116
- BirdCarreau model, U-105
- blended differencing, P-36
- block
 - expansion ratio, U-145
- block keyword, U-143
- blocking
 - keyword entry, U-80
- blockMesh
 - library, U-100
- blockMesh solver, P-45
- blockMesh utility, U-38, U-91, U-142
- blockMesh executable
 - vertex numbering, U-145
- blockMeshDict*
 - dictionary, U-18, U-20, U-36, U-49, U-142, U-151
- blocks keyword, U-20, U-31, U-144
- boundaries, U-135
- boundary, U-135
- boundary*
 - dictionary, U-134, U-142
- boundary keyword, U-147
- boundary condition
 - alphaContactAngle, U-59
 - buoyantPressure, U-141
 - calculated, U-140
 - cyclic, U-139, U-148
 - directionMixed, U-140
 - empty, P-63, P-69, U-18, U-135, U-139
 - fixedGradient, U-140
 - fixedValue, U-140
 - fluxCorrectedVelocity, U-141
 - inlet, P-69
 - inletOutlet, U-141
 - mixed, U-140
 - movingWallVelocity, U-141
 - outlet, P-69
 - outletInlet, U-141
 - partialSlip, U-141
 - patch, U-138
 - pressureDirectedInletVelocity, U-141
 - pressureInletVelocity, U-141
 - pressureOutlet, P-63
 - pressureTransmissive, U-141
 - processor, U-140
 - setup, U-20
 - slip, U-141
 - supersonicFreeStream, U-141
 - surfaceNormalFixedValue, U-141
 - symmetryPlane, P-63, U-139
 - totalPressure, U-141
 - turbulentInlet, U-141
 - wall, U-41
 - wall, P-63, P-69, U-59, U-139
 - wedge, U-135, U-139, U-150
 - zeroGradient, U-140
- boundary conditions, P-41
 - Dirichlet, P-41
 - inlet, P-42
 - Neumann, P-41
 - no-slip impermeable wall, P-42
 - outlet, P-42
 - physical, P-42
 - symmetry plane, P-42
- boundaryField keyword, U-21, U-112
- boundaryFoam solver, U-86
- bounded
 - keyword entry, U-122, U-123
- boxToCell keyword, U-60
- boxTurb utility, U-91
- breaking of a dam, U-56
- BSpline
 - keyword entry, U-144
- buoyantBoussinesqPimpleFoam solver, U-89
- buoyantBoussinesqSimpleFoam solver, U-89
- buoyantPimpleFoam solver, U-89
- buoyantPressure
 - boundary condition, U-141
- buoyantSimpleFoam solver, U-89
- burntProducts keyword, U-195
- button
 - Apply, U-172, U-176
 - Auto Accept, U-176
 - Camera Parallel Projection, U-176
 - Choose Preset, U-175

- Delete, U-172
- Edit Color Map, U-174
- Enable Line Series, U-35
- Lights, U-176
- Orientation Axes, U-24
- Refresh Times, U-25, U-173
- Rescale to Data Range, U-25
- Reset, U-172
- Set Ambient Color, U-175
- Update GUI, U-173
- Use Parallel Projection, U-24

C

C++ syntax

- `/*...*/`, U-78

- `//`, U-78

- `# include`, U-72, U-78

cacheAgglomeration keyword, U-128

calculated

- boundary condition, U-140

cAlpha keyword, U-63

Camera Parallel Projection button, U-176

cases, U-107

castellatedMesh keyword, U-152

castellatedMeshControls

- dictionary, U-154–U-156

castellatedMeshControls keyword, U-152

cavitatingDyMFoam solver, U-87

cavitatingFoam solver, U-87

cavity flow, U-17

ccm26ToFoam utility, U-92

CEI_ARCH

- environment variable, U-186

CEI_HOME

- environment variable, U-186

cell

- expansion ratio, U-145

cell class, P-29

cell

- keyword entry, U-187

cellLimited

- keyword entry, U-122

cellPoint

- keyword entry, U-187

cellPointFace

- keyword entry, U-187

cells

- dictionary, U-142

central differencing, P-36

cfTools tools, U-99

cfx4ToFoam utility, U-92, U-161

changeDictionary utility, U-91

Charts window panel, U-176

checkMesh utility, U-93, U-162

chemFoam solver, U-88

chemistryModel

- library, U-103

chemistryModel model, U-103

chemistrySolver model, U-103

chemkinToFoam utility, U-98

Choose Preset button, U-175

chtMultiRegionSimpleFoam solver, U-89

chtMultiRegionFoam solver, U-89

Chung

- library, U-102

class

- cell, P-29

- dimensionSet, P-24, P-30, P-31

- face, P-29

- finiteVolumeCalculus, P-34

- finiteVolumeMethod, P-34

- fvMesh, P-29

- fvSchemes, P-36

- fvc, P-34

- fvm, P-34

- pointField, P-29

- polyBoundaryMesh, P-29

- polyMesh, P-29, U-131, U-133

- polyPatchList, P-29

- polyPatch, P-29

- scalarField, P-27

- scalar, P-22

- slice, P-29

- symmTensorField, P-27

- symmTensorThirdField, P-27

- tensorField, P-27

- tensorThirdField, P-27

- tensor, P-22

- vectorField, P-27

- vector, P-22, U-111

- word, P-24, P-29

class keyword, U-109

clockTime

- keyword entry, U-116

cloud keyword, U-189

cloudFunctionObjects

- library, U-99

cmptAv

- tensor member function, P-23

Co utility, U-94

coalChemistryFoam solver, U-89

- coalCombustion
 - library, U-100
 - cofactors
 - tensor member function, P-23
 - coldEngineFoam solver, U-88
 - collapseEdges utility, U-94
 - Color By menu, U-175
 - Color Legend window, U-29
 - Color Legend window panel, U-175
 - Color Scale window panel, U-175
 - Colors window panel, U-176
 - compressibleInterDyMFoam solver, U-87
 - compressibleInterFoam solver, U-87
 - compressibleMultiphaseInterFoam solver, U-87
 - combinePatchFaces utility, U-94
 - comments, U-78
 - commsType keyword, U-80
 - compressed
 - keyword entry, U-116
 - compressibleLESModels
 - library, U-105
 - compressibleRASModels
 - library, U-104
 - constant directory, U-107, U-193
 - constant model, U-102
 - constTransport model, U-102
 - containers tools, U-99
 - continuum
 - mechanics, P-13
 - control
 - of time, U-115
 - controlDict
 - dictionary, P-65, U-22, U-31, U-42, U-51, U-62, U-107, U-167
 - controlDict file, P-48
 - convection, *see* divergence, P-36
 - convergence, U-39
 - conversion
 - library, U-100
 - convertToMeters keyword, U-142
 - convertToMeters keyword, U-143
 - coordinate
 - system, P-13
 - coordinate system, U-18
 - corrected
 - keyword entry, U-122, U-123
 - Courant number, P-40, U-22
 - Cp keyword, U-195
 - cpuTime
 - keyword entry, U-116
 - Crank Nicolson
 - temporal discretisation, P-41
 - CrankNicolson
 - keyword entry, U-124
 - createExternalCoupledPatchGeometry utility, U-91
 - createBaffles utility, U-93
 - createPatch utility, U-93
 - createTurbulenceFields utility, U-95
 - cross product, *see* tensor, vector cross product
 - CrossPowerLaw
 - keyword entry, U-60
 - CrossPowerLaw model, U-105
 - cubeRootVolDelta model, U-104
 - cubicCorrected
 - keyword entry, U-124
 - cubicCorrection
 - keyword entry, U-121
 - curl, P-35
 - curl
 - fvc member function, P-35
 - Current Time Controls menu, U-25, U-173
 - curve keyword, U-189
 - Cv keyword, U-195
 - cyclic
 - boundary condition, U-139, U-148
 - cyclic
 - keyword entry, U-139
 - cylinder
 - flow around a, P-43
- ## D
- d2dt2
 - fvc member function, P-35
 - fvm member function, P-35
 - dam
 - breaking of a, U-56
 - datToFoam utility, U-92
 - db tools, U-99
 - ddt
 - fvc member function, P-35
 - fvm member function, P-35
 - DeardorffDiffStress model, U-104, U-105
 - debug keyword, U-152
 - decompose model, U-100
 - decomposePar utility, U-82, U-83, U-98
 - decomposeParDict
 - dictionary, U-82
 - decomposition
 - of field, U-82
 - of mesh, U-82

- decompositionMethods
 - library, U-100
- decompression of a tank, P-61
- defaultFieldValues keyword, U-60
- deformedGeom utility, U-93
- Delete button, U-172
- delta keyword, U-83, U-200
- deltaT keyword, U-116
- dependencies, U-72
- dependency lists, U-72
- det
 - tensor member function, P-23
- determinant, *see* tensor, determinant
- dev
 - tensor member function, P-23
- diag
 - tensor member function, P-23
- diagonal
 - keyword entry, U-126, U-127
- DIC
 - keyword entry, U-127
- DICGaussSeidel
 - keyword entry, U-127
- dictionary
 - LESProperties*, U-199
 - PISO*, U-23
 - blockMeshDict*, U-18, U-20, U-36, U-49, U-142, U-151
 - boundary*, U-134, U-142
 - castellatedMeshControls*, U-154–U-156
 - cells*, U-142
 - controlDict*, P-65, U-22, U-31, U-42, U-51, U-62, U-107, U-167
 - decomposeParDict*, U-82
 - faces*, U-133, U-142
 - fvSchemes*, U-63, U-107, U-118
 - fvSolution*, U-107, U-125
 - mechanicalProperties*, U-51
 - neighbour*, U-134
 - owner*, U-133
 - points*, U-133, U-142
 - thermalProperties*, U-51
 - thermophysicalProperties*, U-193
 - transportProperties*, U-21, U-39, U-42, U-201
 - turbulenceProperties*, U-41, U-61, U-199
- differencing
 - Backward, P-37
 - blended, P-36
 - central, P-36
 - Euler implicit, P-37
 - Gamma, P-36
 - MINMOD, P-36
 - SUPERBEE, P-36
 - upwind, P-36
 - van Leer, P-36
- DILU
 - keyword entry, U-127
- dimension
 - checking in OpenFOAM, P-24, U-111
- dimensional units, U-111
- dimensioned<Type> template class, P-24
- dimensionedTypes tools, U-99
- dimensions keyword, U-21, U-112
- dimensionSet class, P-24, P-30, P-31
- dimensionSet tools, U-99
- directionMixed
 - boundary condition, U-140
- directory
 - 0.000000e+00*, U-108
 - 0*, U-108
 - Make*, U-73
 - constant*, U-107, U-193
 - fluentInterface*, U-183
 - polyMesh*, U-107, U-133
 - processorN*, U-83
 - run*, U-107
 - system*, P-48, U-107
 - tutorials*, P-43, U-17
- discretisation
 - equation, P-31
- Display window panel, U-24, U-25, U-172, U-174
- distance
 - keyword entry, U-156, U-189
- distributed model, U-101
- distributed keyword, U-83, U-84
- distributionModels
 - library, U-100
- div
 - fvc member function, P-35
 - fvm member function, P-35
- divergence, P-35, P-37
- divSchemes keyword, U-118
- dnsFoam solver, U-88
- doLayers keyword, U-152
- double inner product, *see* tensor, double inner product
- DPMFoam solver, U-89
- dsmc
 - library, U-100
- dsmcFieldsCalc utility, U-96

dsmcFoam solver, U-90
 dsmcInitialise utility, U-91
 dx
 keyword entry, U-187
 dynamicFvMesh
 library, U-100
 dynamicMesh
 library, U-100
 dynLagrangian model, U-104
 dynOneEqEddy model, U-104

E

eConstThermo model, U-102
 edgeGrading keyword, U-145
 edgeMesh
 library, U-100
 edges keyword, U-143
 Edit menu, U-176
 Edit Color Map button, U-174
 egrMixture model, U-101
 egrMixture keyword, U-195
 electrostaticFoam solver, U-90
 empty
 boundary condition, P-63, P-69, U-18, U-135, U-139
 empty
 keyword entry, U-139
 Enable Line Series button, U-35
 endTime keyword, U-22, U-115, U-116
 energy keyword, U-194, U-198
 engine
 library, U-100
 engineCompRatio utility, U-96
 engineFoam solver, U-88
 engineSwirl utility, U-91
 ensight74FoamExec utility, U-185
 ENSIGHT7_INPUT
 environment variable, U-186
 ENSIGHT7_READER
 environment variable, U-186
 ensightFoamReader utility, U-94
 enstrophy utility, U-94
 environment variable
 CEI_ARCH, U-186
 CEI_HOME, U-186
 ENSIGHT7_INPUT, U-186
 ENSIGHT7_READER, U-186
 FOAM_RUN, U-107
 WM_ARCH_OPTION, U-76
 WM_ARCH, U-76
 WM_COMPILER_BIN, U-76
 WM_COMPILER_DIR, U-76
 WM_COMPILER_LIB, U-76
 WM_COMPILER, U-76
 WM_COMPILE_OPTION, U-76
 WM_DIR, U-76
 WM_MPLIB, U-76
 WM_OPTIONS, U-76
 WM_PRECISION_OPTION, U-76
 WM_PROJECT_DIR, U-76
 WM_PROJECT_INST_DIR, U-76
 WM_PROJECT_USER_DIR, U-76
 WM_PROJECT_VERSION, U-76
 WM_PROJECT, U-76
 wmake, U-76
 equationOfState keyword, U-194
 equilibriumCO utility, U-98
 equilibriumFlameT utility, U-98
 errorReduction keyword, U-160
 Euler
 keyword entry, U-124
 Euler implicit
 differencing, P-37
 temporal discretisation, P-40
 examples
 decompression of a tank, P-61
 flow around a cylinder, P-43
 flow over backward step, P-50
 Hartmann problem, P-67
 supersonic flow over forward step, P-58
 execFlowFunctionObjects utility, U-96
 expandDictionary utility, U-98
 expansionRatio keyword, U-159
 explicit
 temporal discretisation, P-40
 extrude2DMesh utility, U-92
 extrudeMesh utility, U-91
 extrudeToRegionMesh utility, U-92

F

face class, P-29
 face keyword, U-189
 faceAgglomerate utility, U-91
 faceAreaPair
 keyword entry, U-127
 faceLimited
 keyword entry, U-122
 faces
 dictionary, U-133, U-142
 FDIC
 keyword entry, U-127
 featureAngle keyword, U-159

- features** keyword, U-154
- field**
 - U, U-23
 - p, U-23
 - decomposition, U-82
- FieldField<Type>** template class, P-30
- fieldFunctionObjects**
 - library, U-99
- fields**, P-27
 - mapping, U-167
- fields** tools, U-99
- fields** keyword, U-187
- Field<Type>** template class, P-27
- fieldValues** keyword, U-60
- file**
 - Make/files*, U-75
 - controlDict*, P-48
 - files*, U-73
 - g*, U-60
 - options*, U-73
 - snappyHexMeshDict*, U-152
 - transportProperties*, U-60
- file** format, U-108
- fileFormats**
 - library, U-100
- fileModificationChecking** keyword, U-80
- fileModificationSkew** keyword, U-80
- files** file, U-73
- filteredLinear2**
 - keyword entry, U-121
- finalLayerThickness** keyword, U-159
- financialFoam** solver, U-90
- find** script/alias, U-181
- finite** volume
 - discretisation, P-25
 - mesh, P-29
- finiteVolume**
 - library, U-99
- finiteVolume** tools, U-99
- finiteVolumeCalculus** class, P-34
- finiteVolumeMethod** class, P-34
- fireFoam** solver, U-88
- firstTime** keyword, U-115
- fixed**
 - keyword entry, U-116
- fixedGradient**
 - boundary condition, U-140
- fixedValue**
 - boundary condition, U-140
- flattenMesh** utility, U-93
- floatTransfer** keyword, U-80
- flow**
 - free surface, U-56
 - laminar, U-17
 - steady, turbulent, P-50
 - supersonic, P-58
 - turbulent, U-17
- flow** around a cylinder, P-43
- flow** over backward step, P-50
- flowType** utility, U-94
- fluent3DMeshToFoam** utility, U-92
- fluentInterface** directory, U-183
- fluentMeshToFoam** utility, U-92, U-161
- fluxCorrectedVelocity**
 - boundary condition, U-141
- fluxRequired** keyword, U-118
- OpenFOAM**
 - cases, U-107
- FOAM_RUN**
 - environment variable, U-107
- foamCalc** utility, U-33, U-96
- foamCalcFunctions**
 - library, U-99
- foamChemistryFile** keyword, U-195
- foamCorrectVrt** script/alias, U-166
- foamDataToFluent** utility, U-94, U-183
- foamDebugSwitches** utility, U-98
- FoamFile** keyword, U-109
- foamFile**
 - keyword entry, U-187
- foamFormatConvert** utility, U-98
- foamHelp** utility, U-98
- foamInfoExec** utility, U-98
- foamJob** script/alias, U-190
- foamListTimes** utility, U-96
- foamLog** script/alias, U-190
- foamMeshToFluent** utility, U-92, U-183
- foamToEnlight** utility, U-94
- foamToEnlightParts** utility, U-94
- foamToGMV** utility, U-94
- foamToStarMesh** utility, U-92
- foamToSurface** utility, U-92
- foamToTecplot360** utility, U-94
- foamToVTK** utility, U-94
- foamUpgradeCyclics** utility, U-91
- foamUpgradeFvSolution** utility, U-91
- foamyHexMeshBackgroundMesh** utility, U-92
- foamyHexMeshSurfaceSimplify** utility, U-92
- foamyHexMesh** utility, U-92
- foamyQuadMesh** utility, U-92

forces
 library, U-99
 foreground
 process, U-24
 format keyword, U-109
 fourth
 keyword entry, U-122, U-123
 fuel keyword, U-195
 functionObjectLibs keyword, U-181
 functions keyword, U-117, U-179
 fvc class, P-34
 fvc member function
 curl, P-35
 d2dt2, P-35
 ddt, P-35
 div, P-35
 gGrad, P-35
 grad, P-35
 laplacian, P-35
 lsGrad, P-35
 snGrad, P-35
 snGradCorrection, P-35
 sqrGradGrad, P-35
 fvDOM
 library, U-101
 FVFunctionObjects
 library, U-99
 fvm class, P-34
 fvm member function
 d2dt2, P-35
 ddt, P-35
 div, P-35
 laplacian, P-35
 Su, P-35
 SuSp, P-35
 fvMatrices tools, U-99
 fvMatrix template class, P-34
 fvMesh class, P-29
 fvMesh tools, U-99
 fvMotionSolvers
 library, U-100
 fvSchemes
 dictionary, U-63, U-107, U-118
 fvSchemes class, P-36
 fvSchemes
 menu entry, U-52
 fvSolution
 dictionary, U-107, U-125

G

g file, U-60

gambitToFoam utility, U-92, U-161
 GAMG
 keyword entry, U-53, U-126, U-127
 Gamma
 keyword entry, U-121
 Gamma differencing, P-36
 Gauss
 keyword entry, U-122
 Gauss's theorem, P-34
 GaussSeidel
 keyword entry, U-127
 General window panel, U-176
 general
 keyword entry, U-116
 genericFvPatchField
 library, U-100
 geometric-algebraic multi-grid, U-127
 GeometricBoundaryField template class, P-30
 geometricField<Type> template class, P-30
 geometry keyword, U-152
 gGrad
 fvc member function, P-35
 global tools, U-99
 gmshToFoam utility, U-92
 gnuplot
 keyword entry, U-117, U-187
 grad
 fvc member function, P-35
 (Grad Grad) squared, P-35
 gradient, P-35, P-38
 Gauss scheme, P-38
 Gauss's theorem, U-52
 least square fit, U-52
 least squares method, P-38, U-52
 surface normal, P-38
 gradSchemes keyword, U-118
 graph tools, U-99
 graphFormat keyword, U-117
 GuldersonEGR Laminar Flame Speed model, U-102
 Gulderson Laminar Flame Speed model, U-102

H

hConstThermo model, U-102
 heheuPsiThermo model, U-101
 heheuPsiThermo
 keyword entry, U-194
 Help menu, U-175
 hePsiThermo model, U-101
 hePsiThermo
 keyword entry, U-194
 heRhoThermo model, U-101

heRhoThermo

keyword entry, U-194

HerschelBulkley model, U-105

hExponentialThermo

library, U-103

Hf keyword, U-195

hierarchical

keyword entry, U-82, U-83

highCpCoeffs keyword, U-196

homogenousDynOneEqEddy model, U-104, U-105

homogenousDynSmagorinsky model, U-104

homogeneousMixture model, U-101

homogeneousMixture keyword, U-195

hPolynomialThermo model, U-102

I

I

tensor member function, P-23

icoFoam solver, U-17, U-21, U-22, U-24, U-86

icoPolynomial model, U-102, U-197

icoUncoupledKinematicParcelDyMFoam solver,
U-89

icoUncoupledKinematicParcelFoam solver, U-89

ideasToFoam utility, U-161

ideasUnvToFoam utility, U-92

identities, *see* tensor, identities

identity, *see* tensor, identity

incompressibleLESModels

library, U-104

incompressiblePerfectGas model, U-102, U-197

incompressibleRASModels

library, U-103

incompressibleTransportModels

library, P-53, U-105

incompressibleTurbulenceModels

library, P-53

index

notation, P-14, P-15

Information window panel, U-172

inhomogeneousMixture model, U-101

inhomogeneousMixture keyword, U-195

inlet

boundary condition, P-69

inletOutlet

boundary condition, U-141

inner product, *see* tensor, inner product

inotify

keyword entry, U-80

inotifyMaster

keyword entry, U-80

inside

keyword entry, U-156

insideCells utility, U-93

interPhaseChangeDyMFoam solver, U-88

interPhaseChangeFoam solver, U-88

interDyMFoam solver, U-87

interfaceProperties

library, U-105

interfaceProperties model, U-105

interFoam solver, U-87

interMixingFoam solver, U-87

internalField keyword, U-21, U-112

interpolation tools, U-99

interpolationScheme keyword, U-187

interpolations tools, U-99

interpolationSchemes keyword, U-118

inv

tensor member function, P-23

iterations

maximum, U-126

J

janafThermo model, U-102

jobControl

library, U-99

jplot

keyword entry, U-117, U-187

K

kEpsilon model, U-103, U-104

keyword

As, U-195

Cp, U-195

Cv, U-195

FoamFile, U-109

Hf, U-195

LESModel, U-200

N2, U-195

O2, U-195

Pr, U-195

RASModel, U-200

Tcommon, U-196

Thigh, U-196

Tlow, U-196

Ts, U-195

addLayersControls, U-152

adjustTimeStep, U-62, U-117

agglomerator, U-127

arc, U-143

blocks, U-20, U-31, U-144

block, U-143

boundaryField, U-21, U-112

boundary, U-147
boxToCell, U-60
burntProducts, U-195
cAlpha, U-63
cacheAgglomeration, U-128
castellatedMeshControls, U-152
castellatedMesh, U-152
class, U-109
cloud, U-189
commsType, U-80
convertToMeters, U-143
convertToMeters, U-142
curve, U-189
debug, U-152
defaultFieldValues, U-60
deltaT, U-116
delta, U-83, U-200
dimensions, U-21, U-112
distributed, U-83, U-84
divSchemes, U-118
doLayers, U-152
edgeGrading, U-145
edges, U-143
egrMixture, U-195
endTime, U-22, U-115, U-116
energy, U-194, U-198
equationOfState, U-194
errorReduction, U-160
expansionRatio, U-159
face, U-189
featureAngle, U-159
features, U-154
fieldValues, U-60
fields, U-187
fileModificationChecking, U-80
fileModificationSkew, U-80
finalLayerThickness, U-159
firstTime, U-115
floatTransfer, U-80
fluxRequired, U-118
foamChemistryFile, U-195
format, U-109
fuel, U-195
functionObjectLibs, U-181
functions, U-117, U-179
geometry, U-152
gradSchemes, U-118
graphFormat, U-117
highCpCoeffs, U-196
homogeneousMixture, U-195
inhomogeneousMixture, U-195
internalField, U-21, U-112
interpolationSchemes, U-118
interpolationScheme, U-187
laplacianSchemes, U-118
latestTime, U-39
layers, U-159
leastSquares, U-52
levels, U-156
libs, U-80, U-117
locationInMesh, U-154, U-156
location, U-109
lowCpCoeffs, U-196
manualCoeffs, U-83
maxAlphaCo, U-62
maxBoundarySkewness, U-160
maxConcave, U-160
maxCo, U-62, U-117
maxDeltaT, U-62
maxFaceThicknessRatio, U-159
maxGlobalCells, U-154
maxInternalSkewness, U-160
maxIter, U-126
maxLocalCells, U-154
maxNonOrtho, U-160
maxThicknessToMedialRatio, U-159
mergeLevels, U-128
mergePatchPairs, U-143
mergeTolerance, U-152
meshQualityControls, U-152
method, U-83
midPointAndFace, U-189
midPoint, U-189
minArea, U-160
minDeterminant, U-160
minFaceWeight, U-160
minFlatness, U-160
minMedianAxisAngle, U-159
minRefinementCells, U-154
minThickness, U-159
minTriangleTwist, U-160
minTwist, U-160
minVolRatio, U-160
minVol, U-160
mixture, U-195
mode, U-156
molWeight, U-198
multiComponentMixture, U-195
mu, U-195
nAlphaSubCycles, U-63

- nBufferCellsNoExtrude, U-159
- nCellsBetweenLevels, U-154
- nFaces, U-134
- nFinestSweeps, U-128
- nGrow, U-159
- nLayerIter, U-159
- nMoles, U-198
- nPostSweeps, U-128
- nPreSweeps, U-128
- nRelaxIter, U-157, U-159
- nRelaxedIter, U-159
- nSmoothNormals, U-159
- nSmoothPatch, U-157
- nSmoothScale, U-160
- nSmoothSurfaceNormals, U-159
- nSmoothThickness, U-159
- nSolveIter, U-157
- neighbourPatch, U-148
- numberOfSubdomains, U-83
- nu, U-201
- n, U-83
- object, U-109
- order, U-83
- outputControl, U-181
- oxidant, U-195
- pRefCell, U-23, U-130
- pRefValue, U-23, U-129
- p_rhgRefCell, U-130
- p_rhgRefValue, U-130
- patchMap, U-168
- patches, U-143
- preconditioner, U-126, U-127
- pressure, U-51
- printCoeffs, U-42, U-200
- processorWeights, U-82
- processorWeights, U-83
- purgeWrite, U-116
- refGradient, U-140
- refinementRegions, U-154, U-156
- refinementSurfaces, U-154, U-155
- refinementRegions, U-156
- regions, U-60
- relTol, U-53, U-126
- relativeSizes, U-159
- relaxed, U-160
- resolveFeatureAngle, U-154, U-155
- roots, U-83, U-84
- runTimeModifiable, U-117
- scotchCoeffs, U-83
- setFormat, U-187
- sets, U-187
- simpleGrading, U-145
- simulationType, U-41, U-61, U-199
- singleStepReactingMixture, U-195
- smoother, U-128
- snGradSchemes, U-118
- snapControls, U-152
- snap, U-152
- solvers, U-125
- solver, U-53, U-125
- specie, U-198
- spline, U-143
- startFace, U-134
- startFrom, U-22, U-115
- startTime, U-22, U-115
- stopAt, U-115
- strategy, U-82, U-83
- surfaceFormat, U-187
- surfaces, U-187
- thermoType, U-193
- thermodynamics, U-198
- timeFormat, U-116
- timePrecision, U-117
- timeScheme, U-118
- tolerance, U-53, U-126, U-157
- topoSetSource, U-60
- traction, U-51
- transport, U-194, U-198
- turbulence, U-200
- type, U-137, U-194
- uniform, U-189
- valueFraction, U-140
- value, U-21, U-140
- version, U-109
- vertices, U-20, U-143
- veryInhomogeneousMixture, U-195
- writeCompression, U-116
- writeControl, U-22, U-62, U-116
- writeFormat, U-55, U-116
- writeInterval, U-23, U-32, U-116
- writePrecision, U-116
- <LESModel>Coeffs, U-200
- <RASModel>Coeffs, U-200
- <delta>Coeffs, U-200
- keyword entry
 - BSpline, U-144
 - CrankNicolson, U-124
 - CrossPowerLaw, U-60
 - DICGaussSeidel, U-127
 - DIC, U-127

DILU, U-127
Euler, U-124
FDIC, U-127
GAMG, U-53, U-126, U-127
Gamma, U-121
GaussSeidel, U-127
Gauss, U-122
LESModel, U-41, U-199
MGridGen, U-127
MUSCL, U-121
Newtonian, U-60
PBiCG, U-126
PCG, U-126
QUICK, U-124
RASModel, U-41, U-199
SFCD, U-121, U-124
UMIST, U-119
adjustableRunTime, U-62, U-116
arc, U-144
ascii, U-116
backward, U-124
binary, U-116
blocking, U-80
bounded, U-122, U-123
cellLimited, U-122
cellPointFace, U-187
cellPoint, U-187
cell, U-187
clockTime, U-116
compressed, U-116
corrected, U-122, U-123
cpuTime, U-116
cubicCorrected, U-124
cubicCorrection, U-121
cyclic, U-139
diagonal, U-126, U-127
distance, U-156, U-189
dx, U-187
empty, U-139
faceAreaPair, U-127
faceLimited, U-122
filteredLinear2, U-121
fixed, U-116
foamFile, U-187
fourth, U-122, U-123
general, U-116
gnuplot, U-117, U-187
hePsiThermo, U-194
heRhoThermo, U-194
heheuPsiThermo, U-194
hierarchical, U-82, U-83
inotifyMaster, U-80
inotify, U-80
inside, U-156
jplot, U-117, U-187
laminar, U-41, U-199
latestTime, U-115
leastSquares, U-122
limitedCubic, U-121
limitedLinear, U-121
limited, U-122, U-123
linearUpwind, U-121, U-124
linear, U-121, U-124
line, U-144
localEuler, U-124
manual, U-82, U-83
metis, U-83
midPoint, U-121
nextWrite, U-116
noWriteNow, U-116
nonBlocking, U-80
none, U-119, U-127
null, U-187
outputTime, U-181
outside, U-156
patch, U-139, U-188
polyLine, U-144
processor, U-139
pureMixture, U-195
raw, U-117, U-187
reactingMixture, U-195
runTime, U-32, U-116
scheduled, U-80
scientific, U-116
scotch, U-82, U-83
simple, U-82, U-83
skewLinear, U-121, U-124
smoothSolver, U-126
spline, U-144
startTime, U-22, U-115
steadyState, U-124
stl, U-187
symmetryPlane, U-139
timeStampMaster, U-80
timeStamp, U-80
timeStep, U-23, U-32, U-116, U-181
uncompressed, U-116
uncorrected, U-122, U-123
upwind, U-121, U-124
vanLeer, U-121

- vtk, U-187
 - wall, U-139
 - wedge, U-139
 - writeControl, U-116
 - writeInterval, U-181
 - writeNow, U-115
 - xmgr, U-117, U-187
 - xyz, U-189
 - x, U-189
 - y, U-189
 - z, U-189
 - kivaToFoam utility, U-92
 - kkLOmega model, U-103
 - kOmega model, U-103
 - kOmegaSST model, U-103, U-104
 - kOmegaSSTSAS model, U-104
 - Kronecker delta, P-19
- ## L
- lagrangian
 - library, U-100
 - lagrangianIntermediate
 - library, U-100
 - Lambda2 utility, U-94
 - LamBremhorstKE model, U-103
 - laminar model, U-103, U-104
 - laminar
 - keyword entry, U-41, U-199
 - laminarFlameSpeedModels
 - library, U-102
 - laplaceFilter model, U-104
 - Laplacian, P-36
 - laplacian, P-35
 - laplacian
 - fvc member function, P-35
 - fvm member function, P-35
 - laplacianFoam solver, U-86
 - laplacianSchemes keyword, U-118
 - latestTime
 - keyword entry, U-115
 - latestTime keyword, U-39
 - LaunderGibsonRSTM model, U-103, U-104
 - LaunderSharmaKE model, U-103, U-104
 - layers keyword, U-159
 - leastSquares
 - keyword entry, U-122
 - leastSquares keyword, U-52
 - LESdeltas
 - library, U-104
 - LESfilters
 - library, U-104
 - LESModel
 - keyword entry, U-41, U-199
 - LESModel keyword, U-200
 - LESProperties*
 - dictionary, U-199
 - levels keyword, U-156
 - libraries, U-69
 - library
 - Chung, U-102
 - FVFunctionObjects, U-99
 - LESdeltas, U-104
 - LESfilters, U-104
 - MGridGenGAMGAgglomeration, U-100
 - ODE, U-100
 - OSspecific, U-100
 - OpenFOAM, U-99
 - P1, U-101
 - PV3FoamReader, U-171
 - PVFoamReader, U-171
 - SLGThermo, U-103
 - Wallis, U-102
 - autoMesh, U-100
 - barotropicCompressibilityModels, U-102
 - basicSolidThermo, U-103
 - basicThermophysicalModels, U-101
 - blockMesh, U-100
 - chemistryModel, U-103
 - cloudFunctionObjects, U-99
 - coalCombustion, U-100
 - compressibleLESModels, U-105
 - compressibleRASModels, U-104
 - conversion, U-100
 - decompositionMethods, U-100
 - distributionModels, U-100
 - dsmc, U-100
 - dynamicFvMesh, U-100
 - dynamicMesh, U-100
 - edgeMesh, U-100
 - engine, U-100
 - fieldFunctionObjects, U-99
 - fileFormats, U-100
 - finiteVolume, U-99
 - foamCalcFunctions, U-99
 - forces, U-99
 - fvDOM, U-101
 - fvmotionSolvers, U-100
 - genericFvPatchField, U-100
 - hExponentialThermo, U-103
 - incompressibleLESModels, U-104
 - incompressibleRASModels, U-103

- incompressibleTransportModels, P-53, U-105
 - incompressibleTurbulenceModels, P-53
 - interfaceProperties, U-105
 - jobControl, U-99
 - lagrangianIntermediate, U-100
 - lagrangian, U-100
 - laminarFlameSpeedModels, U-102
 - linear, U-102
 - liquidMixtureProperties, U-103
 - liquidProperties, U-103
 - meshTools, U-100
 - molecularMeasurements, U-100
 - molecule, U-100
 - opaqueSolid, U-102
 - pairPatchAgglomeration, U-100
 - postCalc, U-99
 - potential, U-100
 - primitive, P-21
 - radiationModels, U-101
 - randomProcesses, U-100
 - reactionThermophysicalModels, U-101
 - sampling, U-99
 - solidChemistryModel, U-103
 - solidMixtureProperties, U-103
 - solidParticle, U-100
 - solidProperties, U-103
 - solidSpecie, U-103
 - solidThermo, U-103
 - specie, U-102
 - spray, U-100
 - surfMesh, U-100
 - surfaceFilmModels, U-105
 - systemCall, U-99
 - thermophysicalFunctions, U-102
 - thermophysical, U-193
 - topoChangerFvMesh, U-100
 - triSurface, U-100
 - turbulence, U-100
 - twoPhaseProperties, U-105
 - utilityFunctionObjects, U-99
 - viewFactor, U-102
 - vtkFoam, U-171
 - vtkPV3Foam, U-171
 - libs keyword, U-80, U-117
 - lid-driven cavity flow, U-17
 - LienCubicKE model, U-103
 - LienCubicKELowRe model, U-103
 - LienLeschzinerLowRe model, U-103
 - Lights button, U-176
 - limited
 - keyword entry, U-122, U-123
 - limitedCubic
 - keyword entry, U-121
 - limitedLinear
 - keyword entry, U-121
 - line
 - keyword entry, U-144
 - Line Style menu, U-35
 - linear
 - library, U-102
 - linear model, U-197
 - linear
 - keyword entry, U-121, U-124
 - linearUpwind
 - keyword entry, U-121, U-124
 - liquid
 - electrically-conducting, P-67
 - liquidMixtureProperties
 - library, U-103
 - liquidProperties
 - library, U-103
 - lists, P-27
 - List<Type> template class, P-27
 - localEuler
 - keyword entry, U-124
 - location keyword, U-109
 - locationInMesh keyword, U-154, U-156
 - lowCpCoeffs keyword, U-196
 - lowReOneEqEddy model, U-105
 - LRDDiffStress model, U-104
 - LRR model, U-103, U-104
 - lsGrad
 - fvc member function, P-35
 - LTSInterFoam solver, U-88
 - LTSReactingFoam solver, U-89
 - LTSReactingParcelFoam solver, U-89
- ## M
- Mach utility, U-95
 - mag
 - tensor member function, P-23
 - magneticFoam solver, U-90
 - magnetohydrodynamics, P-67
 - magSqr
 - tensor member function, P-23
 - Make directory, U-73
 - make script/alias, U-71
 - Make/files file, U-75
 - manual
 - keyword entry, U-82, U-83
 - manualCoeffs keyword, U-83

- mapFields utility, U-31, U-38, U-42, U-56, U-91, U-167
- mapping
 - fields, U-167
- Marker Style menu, U-35
- matrices tools, U-99
- max
 - tensor member function, P-23
- maxAlphaCo keyword, U-62
- maxBoundarySkewness keyword, U-160
- maxCo keyword, U-62, U-117
- maxConcave keyword, U-160
- maxDeltaT keyword, U-62
- maxDeltaxyz model, U-104
- maxFaceThicknessRatio keyword, U-159
- maxGlobalCells keyword, U-154
- maximum iterations, U-126
- maxInternalSkewness keyword, U-160
- maxIter keyword, U-126
- maxLocalCells keyword, U-154
- maxNonOrtho keyword, U-160
- maxThicknessToMedialRatio keyword, U-159
- mdEquilibrationFoam solver, U-90
- mdFoam solver, U-90
- mdInitialise utility, U-91
- mechanicalProperties*
 - dictionary, U-51
- memory tools, U-99
- menu
 - Color By, U-175
 - Current Time Controls, U-25, U-173
 - Edit, U-176
 - Help, U-175
 - Line Style, U-35
 - Marker Style, U-35
 - VCR Controls, U-25, U-173
 - View, U-172, U-175
- menu entry
 - Plot Over Line, U-34
 - Save Animation, U-177
 - Save Screenshot, U-177
 - Settings, U-176
 - Solid Color, U-175
 - Toolbars, U-175
 - View Settings, U-24, U-175
 - Wireframe, U-175
 - fvSchemes, U-52
- mergeLevels keyword, U-128
- mergeMeshes utility, U-93
- mergeOrSplitBaffles utility, U-93
- mergePatchPairs keyword, U-143
- mergeTolerance keyword, U-152
- mesh
 - 1-dimensional, U-135
 - 1D, U-135
 - 2-dimensional, U-135
 - 2D, U-135
 - axi-symmetric, U-135
 - basic, P-29
 - block structured, U-142
 - decomposition, U-82
 - description, U-131
 - finite volume, P-29
 - generation, U-142, U-151
 - grading, U-142, U-145
 - grading, example of, P-50
 - non-orthogonal, P-43
 - refinement, P-61
 - resolution, U-29
 - specification, U-131
 - split-hex, U-151
 - Stereolithography (STL), U-151
 - surface, U-151
 - validity constraints, U-131
- Mesh Parts window panel, U-24
- meshes tools, U-99
- meshQualityControls keyword, U-152
- meshTools
 - library, U-100
- message passing interface
 - openMPI, U-84
- method keyword, U-83
- metis
 - keyword entry, U-83
- metisDecomp model, U-101
- MGridGenGAMGAgglomeration
 - library, U-100
- MGridGen
 - keyword entry, U-127
- mhdFoam solver, P-69, U-90
- midPoint
 - keyword entry, U-121
- midPoint keyword, U-189
- midPointAndFace keyword, U-189
- min
 - tensor member function, P-23
- minArea keyword, U-160
- minDeterminant keyword, U-160
- minFaceWeight keyword, U-160
- minFlatness keyword, U-160

- minMedianAxisAngle keyword, U-159
- MINMOD differencing, P-36
- minRefinementCells keyword, U-154
- minThickness keyword, U-159
- minTriangleTwist keyword, U-160
- minTwist keyword, U-160
- minVol keyword, U-160
- minVolRatio keyword, U-160
- mirrorMesh utility, U-93
- mixed
 - boundary condition, U-140
- mixedSmagorinsky model, U-104
- mixture keyword, U-195
- mixtureAdiabaticFlameT utility, U-98
- mode keyword, U-156
- model
 - APIfunctions, U-103
 - BirdCarreau, U-105
 - CrossPowerLaw, U-105
 - DeardorffDiffStress, U-104, U-105
 - GuldersEGRlaminarFlameSpeed, U-102
 - GuldersLaminarFlameSpeed, U-102
 - HerschelBulkley, U-105
 - LRDDiffStress, U-104
 - LRR, U-103, U-104
 - LamBremhorstKE, U-103
 - LaunderGibsonRSTM, U-103, U-104
 - LaunderSharmaKE, U-103, U-104
 - LienCubicKElowRe, U-103
 - LienCubicKE, U-103
 - LienLeschzinerLowRe, U-103
 - NSRDSfunctions, U-102
 - Newtonian, U-105
 - NonlinearKEShah, U-103
 - PengRobinsonGas, U-197
 - PrandtlDelta, U-104
 - RNGkEpsilon, U-103, U-104
 - RaviPetersen, U-102
 - Smagorinsky2, U-104
 - Smagorinsky, U-104, U-105
 - SpalartAllmarasDDES, U-104
 - SpalartAllmarasIDDES, U-104
 - SpalartAllmaras, U-103–U-105
 - adiabaticPerfectFluid, U-102, U-197
 - anisotropicFilter, U-104
 - basicMultiComponentMixture, U-101
 - chemistryModel, U-103
 - chemistrySolver, U-103
 - constTransport, U-102
 - constant, U-102
 - cubeRootVolDelta, U-104
 - decompose, U-100
 - distributed, U-101
 - dynLagrangian, U-104
 - dynOneEqEddy, U-104
 - eConstThermo, U-102
 - egrMixture, U-101
 - hConstThermo, U-102
 - hPolynomialThermo, U-102
 - hePsiThermo, U-101
 - heRhoThermo, U-101
 - heheuPsiThermo, U-101
 - homogenousDynOneEqEddy, U-104, U-105
 - homogenousDynSmagorinsky, U-104
 - homogeneousMixture, U-101
 - icoPolynomial, U-102, U-197
 - incompressiblePerfectGas, U-102, U-197
 - inhomogeneousMixture, U-101
 - interfaceProperties, U-105
 - janafThermo, U-102
 - kEpsilon, U-103, U-104
 - kOmegaSSTAS, U-104
 - kOmegaSST, U-103, U-104
 - kOmega, U-103
 - kkLOmega, U-103
 - laminar, U-103, U-104
 - laplaceFilter, U-104
 - linear, U-197
 - lowReOneEqEddy, U-105
 - maxDeltaxyz, U-104
 - metisDecomp, U-101
 - mixedSmagorinsky, U-104
 - multiComponentMixture, U-101
 - multiphaseMixtureThermo, U-194
 - oneEqEddy, U-104, U-105
 - perfectFluid, U-102, U-197
 - perfectGas, U-197
 - polynomialTransport, U-102
 - powerLaw, U-105
 - psiReactionThermo, U-101, U-194
 - psiThermo, U-194
 - psiuReactionThermo, U-101, U-194
 - ptsotchDecomp, U-101
 - pureMixture, U-101
 - qZeta, U-103
 - reactingMixture, U-101
 - realizableKE, U-103, U-104
 - reconstruct, U-101
 - rhoConst, U-102, U-197
 - rhoReactionThermo, U-101, U-194

- rhoThermo, U-194
- scaleSimilarity, U-104
- scotchDecomp, U-101
- simpleFilter, U-104
- singleStepReactingMixture, U-101
- smoothDelta, U-104
- specieThermo, U-102
- spectEddyVisc, U-104
- sutherlandTransport, U-102
- v2f, U-103, U-104
- vanDriestDelta, U-105
- veryInhomogeneousMixture, U-101
- modifyMesh utility, U-94
- molecularMeasurements
 - library, U-100
- molecule
 - library, U-100
- molWeight keyword, U-198
- moveDynamicMesh utility, U-93
- moveEngineMesh utility, U-93
- moveMesh utility, U-93
- movingWallVelocity
 - boundary condition, U-141
- MPI
 - openMPI, U-84
- MRFFinterFoam solver, U-88
- MRFMultiphaseInterFoam solver, U-88
- mshToFoam utility, U-92
- mu keyword, U-195
- multiComponentMixture model, U-101
- multiComponentMixture keyword, U-195
- multigrid
 - geometric-algebraic, U-127
- multiphaseEulerFoam solver, U-88
- multiphaseInterFoam solver, U-88
- multiphaseMixtureThermo model, U-194
- MUSCL
 - keyword entry, U-121

N

- n keyword, U-83
- N2 keyword, U-195
- nabla
 - operator, P-25
- nAlphaSubCycles keyword, U-63
- nBufferCellsNoExtrude keyword, U-159
- nCellsBetweenLevels keyword, U-154
- neighbour
 - dictionary, U-134
- neighbourPatch keyword, U-148
- netgenNeutralToFoam utility, U-92

- Newtonian
 - keyword entry, U-60
- Newtonian model, U-105
- nextWrite
 - keyword entry, U-116
- nFaces keyword, U-134
- nFinestSweeps keyword, U-128
- nGrow keyword, U-159
- nLayerIter keyword, U-159
- nMoles keyword, U-198
- non-orthogonal mesh, P-43
- nonBlocking
 - keyword entry, U-80
- none
 - keyword entry, U-119, U-127
- NonlinearKEShik model, U-103
- nonNewtonianIcoFoam solver, U-86
- noWriteNow
 - keyword entry, U-116
- nPostSweeps keyword, U-128
- nPreSweeps keyword, U-128
- nRelaxedIter keyword, U-159
- nRelaxIter keyword, U-157, U-159
- nSmoothNormals keyword, U-159
- nSmoothPatch keyword, U-157
- nSmoothScale keyword, U-160
- nSmoothSurfaceNormals keyword, U-159
- nSmoothThickness keyword, U-159
- nSolveIter keyword, U-157
- NSRDSfunctions model, U-102
- nu keyword, U-201
- null
 - keyword entry, U-187
- numberOfSubdomains keyword, U-83

O

- O2 keyword, U-195
- object keyword, U-109
- objToVTK utility, U-93
- ODE
 - library, U-100
- oneEqEddy model, U-104, U-105
- Opacity text box, U-175
- opaqueSolid
 - library, U-102
- OpenFOAM
 - applications, U-69
 - file format, U-108
 - libraries, U-69
- OpenFOAM
 - library, U-99

OpenFOAM file syntax
 //, U-108
 openMPI
 message passing interface, U-84
 MPI, U-84
 operator
 scalar, P-26
 vector, P-25
 Options window, U-176
options file, U-73
 order keyword, U-83
 Orientation Axes button, U-24
 orientFaceZone utility, U-93
 OSspecific
 library, U-100
 outer product, *see* tensor, outer product
 outlet
 boundary condition, P-69
 outletInlet
 boundary condition, U-141
 outputControl keyword, U-181
 outputTime
 keyword entry, U-181
 outside
 keyword entry, U-156
 owner
 dictionary, U-133
 oxidant keyword, U-195

P

p field, U-23
 P1
 library, U-101
 p_rhgRefCell keyword, U-130
 p_rhgRefValue keyword, U-130
 pairPatchAgglomeration
 library, U-100
 paraFoam, U-23, U-171
 parallel
 running, U-81
 Paramters window panel, U-173
 partialSlip
 boundary condition, U-141
 particleTracks utility, U-95
 patch
 boundary condition, U-138
 patch
 keyword entry, U-139, U-188
 patchAverage utility, U-95
 patches keyword, U-143
 patchIntegrate utility, U-95
 patchMap keyword, U-168
 patchSummary utility, U-98
 PBiCG
 keyword entry, U-126
 PCG
 keyword entry, U-126
 pdfPlot utility, U-96
 PDRFoam solver, U-89
 PDRMesh utility, U-94
 Pe utility, U-95
 PengRobinsonGas model, U-197
 perfectFluid model, U-102, U-197
 perfectGas model, U-197
 permutation symbol, P-18
 pimpleDyMFoam solver, U-86
 pimpleFoam solver, U-86
 Pipeline Browser window, U-24, U-172
PISO
 dictionary, U-23
 pisoFoam solver, U-17, U-86
 Plot Over Line
 menu entry, U-34
 plot3dToFoam utility, U-92
 pointField class, P-29
 pointField<Type> template class, P-31
points
 dictionary, U-133, U-142
 polyBoundaryMesh class, P-29
 polyDualMesh utility, U-93
 polyLine
 keyword entry, U-144
polyMesh directory, U-107, U-133
 polyMesh class, P-29, U-131, U-133
 polynomialTransport model, U-102
 polyPatch class, P-29
 polyPatchList class, P-29
 porousInterFoam solver, U-88
 porousSimpleFoam solver, U-86
 post-processing, U-171
 post-processing
 paraFoam, U-171
 postCalc
 library, U-99
 postChannel utility, U-96
 potentialFreeSurfaceFoam solver, U-88
 potential
 library, U-100
 potentialFoam solver, P-44, U-86
 pow
 tensor member function, P-23

powerLaw model, U-105
pPrime2 utility, U-95
Pr keyword, U-195
PrandtlDelta model, U-104
preconditioner keyword, U-126, U-127
pRefCell keyword, U-23, U-130
pRefValue keyword, U-23, U-129
pressure keyword, U-51
pressure waves
 in liquids, P-62
pressureDirectedInletVelocity
 boundary condition, U-141
pressureInletVelocity
 boundary condition, U-141
pressureOutlet
 boundary condition, P-63
pressureTransmissive
 boundary condition, U-141
primitive
 library, P-21
primitives tools, U-99
printCoeffs keyword, U-42, U-200
processorWeights keyword, U-82
probeLocations utility, U-96
process
 background, U-24, U-81
 foreground, U-24
processor
 boundary condition, U-140
processor
 keyword entry, U-139
processorN directory, U-83
processorWeights keyword, U-83
Properties window, U-173, U-174
Properties window panel, U-25, U-172
psiReactionThermo model, U-101, U-194
psiThermo model, U-194
psiuReactionThermo model, U-101, U-194
ptot utility, U-96
ptsotchDecomp model, U-101
pureMixture model, U-101
pureMixture
 keyword entry, U-195
purgeWrite keyword, U-116
PV3FoamReader
 library, U-171
PVFoamReader
 library, U-171

Q

Q utility, U-95

QUICK
 keyword entry, U-124
qZeta model, U-103

R

R utility, U-95
radiationModels
 library, U-101
randomProcesses
 library, U-100
RASModel
 keyword entry, U-41, U-199
RASModel keyword, U-200
RaviPetersen model, U-102
raw
 keyword entry, U-117, U-187
reactingFoam solver, U-89
reactingMixture model, U-101
reactingMixture
 keyword entry, U-195
reactingParcelFilmFoam solver, U-90
reactingParcelFoam solver, U-90
reactionThermophysicalModels
 library, U-101
realizableKE model, U-103, U-104
reconstruct model, U-101
reconstructPar utility, U-85
reconstructParMesh utility, U-98
redistributePar utility, U-98
refGradient keyword, U-140
refineHexMesh utility, U-94
refinementRegions keyword, U-156
refinementLevel utility, U-94
refinementRegions keyword, U-154, U-156
refinementSurfaces keyword, U-154, U-155
refineMesh utility, U-93
refineWallLayer utility, U-94
Refresh Times button, U-25, U-173
regions keyword, U-60
relative tolerance, U-126
relativeSizes keyword, U-159
relaxed keyword, U-160
relTol keyword, U-53, U-126
removeFaces utility, U-94
Render View window, U-176
Render View window panel, U-175, U-176
renumberMesh utility, U-93
Rescale to Data Range button, U-25
Reset button, U-172
resolveFeatureAngle keyword, U-154, U-155
restart, U-39

Reynolds number, U-17, U-21
 rhoPorousSimpleFoam solver, U-87
 rhoReactingBuoyantFoam solver, U-89
 rhoCentralDyMFoam solver, U-86
 rhoCentralFoam solver, U-86
 rhoConst model, U-102, U-197
 rhoLTSPimpleFoam solver, U-86
 rhoPimpleFoam solver, U-87
 rhoPimplecFoam solver, U-87
 rhoReactingFoam solver, U-89
 rhoReactionThermo model, U-101, U-194
 rhoSimpleFoam solver, U-87
 rhoSimplecFoam solver, U-87
 rhoThermo model, U-194
 rmdepall script/alias, U-77
 RNGkEpsilon model, U-103, U-104
 roots keyword, U-83, U-84
 rotateMesh utility, U-93
 run
 parallel, U-81
 run directory, U-107
 runTime
 keyword entry, U-32, U-116
 runTimeModifiable keyword, U-117

S

sammToFoam utility, U-92
 sample utility, U-96, U-186
 sampling
 library, U-99
 Save Animation
 menu entry, U-177
 Save Screenshot
 menu entry, U-177
 scalar, P-14
 operator, P-26
 scalar class, P-22
 scalarField class, P-27
 scalarTransportFoam solver, U-86
 scale
 tensor member function, P-23
 scalePoints utility, U-164
 scaleSimilarity model, U-104
 scheduled
 keyword entry, U-80
 scientific
 keyword entry, U-116
 scotch
 keyword entry, U-82, U-83
 scotchCoeffs keyword, U-83
 scotchDecomp model, U-101
 script/alias
 find, U-181
 foamCorrectVrt, U-166
 foamJob, U-190
 foamLog, U-190
 make, U-71
 rmdepall, U-77
 wclean, U-76
 wmake, U-71
 second time derivative, P-35
 Seed window, U-177
 selectCells utility, U-94
 Set Ambient Color button, U-175
 setFields utility, U-60, U-91
 setFormat keyword, U-187
 sets keyword, U-187
 setSet utility, U-93
 setsToZones utility, U-93
 Settings
 menu entry, U-176
 settlingFoam solver, U-88
 SFCD
 keyword entry, U-121, U-124
 shallowWaterFoam solver, U-86
 shape, U-145
 SI units, U-112
 simpleReactingParcelFoam solver, U-90
 simple
 keyword entry, U-82, U-83
 simpleFilter model, U-104
 simpleFoam solver, P-53, U-86
 simpleGrading keyword, U-145
 simulationType keyword, U-41, U-61, U-199
 singleCellMesh utility, U-93
 singleStepReactingMixture model, U-101
 singleStepReactingMixture keyword, U-195
 skew
 tensor member function, P-23
 skewLinear
 keyword entry, U-121, U-124
 SLGThermo
 library, U-103
 slice class, P-29
 slip
 boundary condition, U-141
 Smagorinsky model, U-104, U-105
 Smagorinsky2 model, U-104
 smapToFoam utility, U-94
 smoothDelta model, U-104
 smoother keyword, U-128

- `smoothSolver`
 - keyword entry, U-126
- `snap` keyword, U-152
- `snapControls` keyword, U-152
- `snappyHexMesh` utility
 - background mesh, U-153
 - cell removal, U-156
 - cell splitting, U-154
 - mesh layers, U-157
 - meshing process, U-151
 - snapping to surfaces, U-157
- `snappyHexMesh` utility, U-92, U-151
- `snappyHexMeshDict` file, U-152
- `snGrad`
 - fvc member function, P-35
- `snGradCorrection`
 - fvc member function, P-35
- `snGradSchemes` keyword, U-118
- `Solid Color`
 - menu entry, U-175
- `solidChemistryModel`
 - library, U-103
- `solidDisplacementFoam` solver, U-90
- `solidDisplacementFoam` solver, U-51
- `solidEquilibriumDisplacementFoam` solver, U-90
- `solidMixtureProperties`
 - library, U-103
- `solidParticle`
 - library, U-100
- `solidProperties`
 - library, U-103
- `solidSpecie`
 - library, U-103
- `solidThermo`
 - library, U-103
- `solver`
 - DPMFoam, U-89
 - LTSInterFoam, U-88
 - LTSReactingFoam, U-89
 - LTSReactingParcelFoam, U-89
 - MRFInterFoam, U-88
 - MRFMultiphaseInterFoam, U-88
 - PDRFoam, U-89
 - SRFPimpleFoam, U-86
 - SRFSimpleFoam, U-86
 - XiFoam, U-89
 - adjointShapeOptimizationFoam, U-86
 - blockMesh, P-45
 - boundaryFoam, U-86
 - buoyantBoussinesqPimpleFoam, U-89
 - buoyantBoussinesqSimpleFoam, U-89
 - buoyantPimpleFoam, U-89
 - buoyantSimpleFoam, U-89
 - cavitatingDyMFoam, U-87
 - cavitatingFoam, U-87
 - chemFoam, U-88
 - chtMultiRegionFoam, U-89
 - chtMultiRegionSimpleFoam, U-89
 - coalChemistryFoam, U-89
 - coldEngineFoam, U-88
 - compressibleInterDyMFoam, U-87
 - compressibleInterFoam, U-87
 - compressibleMultiphaseInterFoam, U-87
 - dnsFoam, U-88
 - dsmcFoam, U-90
 - electrostaticFoam, U-90
 - engineFoam, U-88
 - financialFoam, U-90
 - fireFoam, U-88
 - icoFoam, U-17, U-21, U-22, U-24, U-86
 - icoUncoupledKinematicParcelDyMFoam, U-89
 - icoUncoupledKinematicParcelFoam, U-89
 - interDyMFoam, U-87
 - interFoam, U-87
 - interMixingFoam, U-87
 - interPhaseChangeDyMFoam, U-88
 - interPhaseChangeFoam, U-88
 - laplacianFoam, U-86
 - magneticFoam, U-90
 - mdEquilibrationFoam, U-90
 - mdFoam, U-90
 - mhdFoam, P-69, U-90
 - multiphaseEulerFoam, U-88
 - multiphaseInterFoam, U-88
 - nonNewtonianIcoFoam, U-86
 - pimpleDyMFoam, U-86
 - pimpleFoam, U-86
 - pisoFoam, U-17, U-86
 - porousInterFoam, U-88
 - porousSimpleFoam, U-86
 - potentialFreeSurfaceFoam, U-88
 - potentialFoam, P-44, U-86
 - reactingFoam, U-89
 - reactingParcelFilmFoam, U-90
 - reactingParcelFoam, U-90
 - rhoCentralDyMFoam, U-86
 - rhoCentralFoam, U-86
 - rhoLTSPimpleFoam, U-86
 - rhoPimpleFoam, U-87

- rhoPimplecFoam, U-87
- rhoReactingFoam, U-89
- rhoSimpleFoam, U-87
- rhoSimplecFoam, U-87
- rhoPorousSimpleFoam, U-87
- rhoReactingBuoyantFoam, U-89
- scalarTransportFoam, U-86
- settlingFoam, U-88
- shallowWaterFoam, U-86
- simpleReactingParcelFoam, U-90
- simpleFoam, P-53, U-86
- solidDisplacementFoam, U-90
- solidDisplacementFoam, U-51
- solidEquilibriumDisplacementFoam, U-90
- sonicDyMFoam, U-87
- sonicFoam, P-59, U-87
- sonicLiquidFoam, P-63, U-87
- sprayEngineFoam, U-90
- sprayFoam, U-90
- thermoFoam, U-89
- twoLiquidMixingFoam, U-88
- twoPhaseEulerFoam, U-88
- uncoupledKinematicParcelFoam, U-90
- solver keyword, U-53, U-125
- solver relative tolerance, U-126
- solver tolerance, U-126
- solvers keyword, U-125
- sonicDyMFoam solver, U-87
- sonicFoam solver, P-59, U-87
- sonicLiquidFoam solver, P-63, U-87
- source, P-35
- SpalartAllmaras model, U-103–U-105
- SpalartAllmarasDDES model, U-104
- SpalartAllmarasIDDES model, U-104
- specie
 - library, U-102
- specie keyword, U-198
- specieThermo model, U-102
- spectEddyVisc model, U-104
- spline
 - keyword entry, U-144
- spline keyword, U-143
- splitCells utility, U-94
- splitMesh utility, U-93
- splitMeshRegions utility, U-93
- spray
 - library, U-100
- sprayEngineFoam solver, U-90
- sprayFoam solver, U-90
- sqr
 - tensor member function, P-23
- sqrGradGrad
 - fvc member function, P-35
- SRFPimpleFoam solver, U-86
- SRFSimpleFoam solver, U-86
- star3ToFoam utility, U-92
- star4ToFoam utility, U-92
- startFace keyword, U-134
- startFrom keyword, U-22, U-115
- starToFoam utility, U-161
- startTime
 - keyword entry, U-22, U-115
- startTime keyword, U-22, U-115
- steady flow
 - turbulent, P-50
- steadyParticleTracks utility, U-96
- steadyState
 - keyword entry, U-124
- Stereolithography (STL), U-151
- stitchMesh utility, U-93
- stl
 - keyword entry, U-187
- stopAt keyword, U-115
- strategy keyword, U-82, U-83
- streamFunction utility, U-95
- stress analysis of plate with hole, U-46
- stressComponents utility, U-95
- Style window panel, U-175
- Su
 - fvm member function, P-35
- subsetMesh utility, U-93
- summation convention, P-15
- SUPERBEE differencing, P-36
- supersonic flow, P-58
- supersonic flow over forward step, P-58
- supersonicFreeStream
 - boundary condition, U-141
- surfaceLambdaMuSmooth utility, U-97
- surface mesh, U-151
- surfaceAdd utility, U-96
- surfaceAutoPatch utility, U-96
- surfaceBooleanFeatures utility, U-96
- surfaceCheck utility, U-96
- surfaceClean utility, U-96
- surfaceCoarsen utility, U-96
- surfaceConvert utility, U-96
- surfaceFeatureConvert utility, U-96
- surfaceFeatureExtract utility, U-96, U-155
- surfaceField<Type> template class, P-31
- surfaceFilmModels

- library, U-105
- surfaceFind utility, U-96
- surfaceFormat keyword, U-187
- surfaceHookUp utility, U-96
- surfaceInertia utility, U-97
- surfaceMesh tools, U-99
- surfaceMeshConvert utility, U-97
- surfaceMeshConvertTesting utility, U-97
- surfaceMeshExport utility, U-97
- surfaceMeshImport utility, U-97
- surfaceMeshInfo utility, U-97
- surfaceMeshTriangulate utility, U-97
- surfaceNormalFixedValue
 - boundary condition, U-141
- surfaceOrient utility, U-97
- surfacePointMerge utility, U-97
- surfaceRedistributePar utility, U-97
- surfaceRefineRedGreen utility, U-97
- surfaces keyword, U-187
- surfaceSplitByPatch utility, U-97
- surfaceSplitByTopology utility, U-97
- surfaceSplitNonManifolds utility, U-97
- surfaceSubset utility, U-97
- surfaceToPatch utility, U-97
- surfaceTransformPoints utility, U-97
- surfMesh
 - library, U-100
- SuSp
 - fvm member function, P-35
- sutherlandTransport model, U-102
- symm
 - tensor member function, P-23
- symmetryPlane
 - boundary condition, P-63, U-139
- symmetryPlane
 - keyword entry, U-139
- symmTensorField class, P-27
- symmTensorThirdField class, P-27
- system directory, P-48, U-107
- systemCall
 - library, U-99

T

- T()
 - tensor member function, P-23
- Tcommon keyword, U-196
- template class
 - GeometricBoundaryField, P-30
 - fvMatrix, P-34
 - dimensioned<Type>, P-24
 - FieldField<Type>, P-30

- Field<Type>, P-27
- geometricField<Type>, P-30
- List<Type>, P-27
- pointField<Type>, P-31
- surfaceField<Type>, P-31
- volField<Type>, P-31
- temporal discretisation, P-40
 - Crank Nicolson, P-41
 - Euler implicit, P-40
 - explicit, P-40
 - in OpenFOAM, P-41
- temporalInterpolate utility, U-96
- tensor, P-13
 - addition, P-16
 - algebraic operations, P-16
 - algebraic operations in OpenFOAM, P-22
 - antisymmetric, *see* tensor, skew
 - calculus, P-25
 - classes in OpenFOAM, P-21
 - cofactors, P-20
 - component average, P-18
 - component maximum, P-18
 - component minimum, P-18
 - determinant, P-20
 - deviatoric, P-20
 - diagonal, P-20
 - dimension, P-14
 - double inner product, P-17
 - geometric transformation, P-19
 - Hodge dual, P-21
 - hydrostatic, P-20
 - identities, P-19
 - identity, P-19
 - inner product, P-16
 - inverse, P-21
 - magnitude, P-18
 - magnitude squared, P-18
 - mathematics, P-13
 - notation, P-15
 - n*th power, P-18
 - outer product, P-17
 - rank, P-14
 - rank 3, P-15
 - scalar division, P-16
 - scalar multiplication, P-16
 - scale function, P-18
 - second rank, P-14
 - skew, P-20
 - square of, P-18
 - subtraction, P-16

- symmetric, P-20
- symmetric rank 2, P-14
- symmetric rank 3, P-15
- trace, P-20
- transformation, P-19
- transpose, P-14, P-20
- triple inner product, P-17
- vector cross product, P-18
- tensor class, P-22
- tensor member function
 - `*`, P-23
 - `+`, P-23
 - `-`, P-23
 - `/`, P-23
 - `&`, P-23
 - `&&`, P-23
 - `^`, P-23
 - `cmptAv`, P-23
 - `cofactors`, P-23
 - `det`, P-23
 - `dev`, P-23
 - `diag`, P-23
 - `I`, P-23
 - `inv`, P-23
 - `mag`, P-23
 - `magSqr`, P-23
 - `max`, P-23
 - `min`, P-23
 - `pow`, P-23
 - `scale`, P-23
 - `skew`, P-23
 - `sqr`, P-23
 - `symm`, P-23
 - `T()`, P-23
 - `tr`, P-23
 - `transform`, P-23
- tensorField class, P-27
- tensorThirdField class, P-27
- tetgenToFoam utility, U-92
- text box
 - Opacity, U-175
- thermalProperties*
 - dictionary, U-51
- thermodynamics keyword, U-198
- thermoFoam solver, U-89
- thermophysical
 - library, U-193
- thermophysicalFunctions
 - library, U-102
- thermophysicalProperties*
 - dictionary, U-193
- thermoType keyword, U-193
- Thigh keyword, U-196
- time
 - control, U-115
- time derivative, P-35
 - first, P-37
 - second, P-35, P-37
- time step, U-22
- timeFormat keyword, U-116
- timePrecision keyword, U-117
- timeScheme keyword, U-118
- timeStamp
 - keyword entry, U-80
- timeStampMaster
 - keyword entry, U-80
- timeStep
 - keyword entry, U-23, U-32, U-116, U-181
- Tlow keyword, U-196
- tolerance
 - solver, U-126
 - solver relative, U-126
- tolerance keyword, U-53, U-126, U-157
- Toolbars
 - menu entry, U-175
- tools
 - algorithms, U-99
 - cfTools, U-99
 - containers, U-99
 - db, U-99
 - dimensionSet, U-99
 - dimensionedTypes, U-99
 - fields, U-99
 - finiteVolume, U-99
 - fvMatrices, U-99
 - fvMesh, U-99
 - global, U-99
 - graph, U-99
 - interpolations, U-99
 - interpolation, U-99
 - matrices, U-99
 - memory, U-99
 - meshes, U-99
 - primitives, U-99
 - surfaceMesh, U-99
 - volMesh, U-99
- topoChangerFvMesh
 - library, U-100
- topoSet utility, U-93
- topoSetSource keyword, U-60

- totalPressure
 - boundary condition, U-141
 - tr
 - tensor member function, P-23
 - trace, *see* tensor, trace
 - traction keyword, U-51
 - transform
 - tensor member function, P-23
 - transformPoints utility, U-94
 - transport keyword, U-194, U-198
 - transportProperties*
 - dictionary, U-21, U-39, U-42, U-201
 - transportProperties* file, U-60
 - triple inner product, P-17
 - triSurface
 - library, U-100
 - Ts keyword, U-195
 - turbulence
 - dissipation, U-40
 - kinetic energy, U-40
 - length scale, U-41
 - turbulence
 - library, U-100
 - turbulence keyword, U-200
 - turbulence model
 - RAS, U-40
 - turbulenceProperties*
 - dictionary, U-41, U-61, U-199
 - turbulent flow
 - steady, P-50
 - turbulentInlet
 - boundary condition, U-141
 - tutorials
 - breaking of a dam, U-56
 - lid-driven cavity flow, U-17
 - stress analysis of plate with hole, U-46
 - tutorials* directory, P-43, U-17
 - twoLiquidMixingFoam solver, U-88
 - twoPhaseEulerFoam solver, U-88
 - twoPhaseProperties
 - library, U-105
 - type keyword, U-137, U-194
- U**
- U field, U-23
 - Ucomponents utility, P-70
 - UMIST
 - keyword entry, U-119
 - uncompressed
 - keyword entry, U-116
 - uncorrected
 - keyword entry, U-122, U-123
 - uncoupledKinematicParcelFoam solver, U-90
 - uniform keyword, U-189
 - units
 - base, U-112
 - of measurement, P-24, U-111
 - S.I. base, P-24
 - SI, U-112
 - Système International, U-112
 - United States Customary System, U-112
 - USCS, U-112
 - Update GUI button, U-173
 - uprime utility, U-95
 - upwind
 - keyword entry, U-121, U-124
 - upwind differencing, P-36, U-63
 - USCS units, U-112
 - Use Parallel Projection button, U-24
 - utility
 - Co, U-94
 - Lambda2, U-94
 - Mach, U-95
 - PDRMesh, U-94
 - Pe, U-95
 - Q, U-95
 - R, U-95
 - Ucomponents, P-70
 - adiabaticFlameT, U-98
 - ansysToFoam, U-92
 - applyBoundaryLayer, U-91
 - applyWallFunctionBoundaryConditions, U-91
 - attachMesh, U-93
 - autoPatch, U-93
 - autoRefineMesh, U-94
 - blockMesh, U-38, U-91, U-142
 - boxTurb, U-91
 - ccm26ToFoam, U-92
 - cfx4ToFoam, U-92, U-161
 - changeDictionary, U-91
 - checkMesh, U-93, U-162
 - chemkinToFoam, U-98
 - collapseEdges, U-94
 - combinePatchFaces, U-94
 - createBaffles, U-93
 - createPatch, U-93
 - createTurbulenceFields, U-95
 - createExternalCoupledPatchGeometry, U-91
 - datToFoam, U-92
 - decomposePar, U-82, U-83, U-98
 - deformedGeom, U-93

- dsmcFieldsCalc, U-96
- dsmcInitialise, U-91
- engineCompRatio, U-96
- engineSwirl, U-91
- ensight74FoamExec, U-185
- ensightFoamReader, U-94
- enstrophy, U-94
- equilibriumCO, U-98
- equilibriumFlameT, U-98
- execFlowFunctionObjects, U-96
- expandDictionary, U-98
- extrude2DMesh, U-92
- extrudeMesh, U-91
- extrudeToRegionMesh, U-92
- faceAgglomerate, U-91
- flattenMesh, U-93
- flowType, U-94
- fluent3DMeshToFoam, U-92
- fluentMeshToFoam, U-92, U-161
- foamCalc, U-33, U-96
- foamDataToFluent, U-94, U-183
- foamDebugSwitches, U-98
- foamFormatConvert, U-98
- foamHelp, U-98
- foamInfoExec, U-98
- foamListTimes, U-96
- foamMeshToFluent, U-92, U-183
- foamToEnightParts, U-94
- foamToEnight, U-94
- foamToGMV, U-94
- foamToStarMesh, U-92
- foamToSurface, U-92
- foamToTecplot360, U-94
- foamToVTK, U-94
- foamUpgradeCyclics, U-91
- foamUpgradeFvSolution, U-91
- foamyHexMesh, U-92
- foamyQuadMesh, U-92
- foamyHexMeshBackgroundMesh, U-92
- foamyHexMeshSurfaceSimplify, U-92
- gambitToFoam, U-92, U-161
- gmshToFoam, U-92
- ideasToFoam, U-161
- ideasUnvToFoam, U-92
- insideCells, U-93
- kivaToFoam, U-92
- mapFields, U-31, U-38, U-42, U-56, U-91, U-167
- mdInitialise, U-91
- mergeMeshes, U-93
- mergeOrSplitBaffles, U-93
- mirrorMesh, U-93
- mixtureAdiabaticFlameT, U-98
- modifyMesh, U-94
- moveDynamicMesh, U-93
- moveEngineMesh, U-93
- moveMesh, U-93
- mshToFoam, U-92
- netgenNeutralToFoam, U-92
- objToVTK, U-93
- orientFaceZone, U-93
- pPrime2, U-95
- particleTracks, U-95
- patchAverage, U-95
- patchIntegrate, U-95
- patchSummary, U-98
- pdfPlot, U-96
- plot3dToFoam, U-92
- polyDualMesh, U-93
- postChannel, U-96
- probeLocations, U-96
- ptot, U-96
- reconstructParMesh, U-98
- reconstructPar, U-85
- redistributePar, U-98
- refineHexMesh, U-94
- refineMesh, U-93
- refineWallLayer, U-94
- refinementLevel, U-94
- removeFaces, U-94
- renumberMesh, U-93
- rotateMesh, U-93
- sammToFoam, U-92
- sample, U-96, U-186
- scalePoints, U-164
- selectCells, U-94
- setFields, U-60, U-91
- setSet, U-93
- setsToZones, U-93
- singleCellMesh, U-93
- smapToFoam, U-94
- snappyHexMesh, U-92, U-151
- splitCells, U-94
- splitMeshRegions, U-93
- splitMesh, U-93
- star3ToFoam, U-92
- star4ToFoam, U-92
- starToFoam, U-161
- steadyParticleTracks, U-96
- stitchMesh, U-93

- streamFunction, U-95
- stressComponents, U-95
- subsetMesh, U-93
- surfaceLambdaMuSmooth, U-97
- surfaceAdd, U-96
- surfaceAutoPatch, U-96
- surfaceBooleanFeatures, U-96
- surfaceCheck, U-96
- surfaceClean, U-96
- surfaceCoarsen, U-96
- surfaceConvert, U-96
- surfaceFeatureConvert, U-96
- surfaceFeatureExtract, U-96, U-155
- surfaceFind, U-96
- surfaceHookUp, U-96
- surfaceInertia, U-97
- surfaceMeshConvertTesting, U-97
- surfaceMeshConvert, U-97
- surfaceMeshExport, U-97
- surfaceMeshImport, U-97
- surfaceMeshInfo, U-97
- surfaceMeshTriangulate, U-97
- surfaceOrient, U-97
- surfacePointMerge, U-97
- surfaceRedistributePar, U-97
- surfaceRefineRedGreen, U-97
- surfaceSplitByPatch, U-97
- surfaceSplitByTopology, U-97
- surfaceSplitNonManifolds, U-97
- surfaceSubset, U-97
- surfaceToPatch, U-97
- surfaceTransformPoints, U-97
- temporalInterpolate, U-96
- tetgenToFoam, U-92
- topoSet, U-93
- transformPoints, U-94
- uprime, U-95
- viewFactorsGen, U-91
- vorticity, U-95
- vtkUnstructuredToFoam, U-92
- wallFunctionTable, U-91
- wallGradU, U-95
- wallHeatFlux, U-95
- wallShearStress, U-95
- wdot, U-96
- writeCellCentres, U-96
- writeMeshObj, U-92
- yPlusLES, U-95
- yPlusRAS, U-95
- zipUpMesh, U-94

- utilityFunctionObjects
 - library, U-99

V

- v2f model, U-103, U-104
- value keyword, U-21, U-140
- valueFraction keyword, U-140
- van Leer differencing, P-36
- vanDriestDelta model, U-105
- vanLeer
 - keyword entry, U-121
- VCR Controls menu, U-25, U-173
- vector, P-14
 - operator, P-25
 - unit, P-18
- vector class, P-22, U-111
- vector product, *see* tensor, vector cross product
- vectorField class, P-27
- version keyword, U-109
- vertices keyword, U-20, U-143
- veryInhomogeneousMixture model, U-101
- veryInhomogeneousMixture keyword, U-195
- View menu, U-172, U-175
- View Render window panel, U-24
- View Settings
 - menu entry, U-24, U-175
- viewFactor
 - library, U-102
- viewFactorsGen utility, U-91
- viscosity
 - kinematic, U-22, U-42
- volField<Type> template class, P-31
- volMesh tools, U-99
- vorticity utility, U-95
- vtk
 - keyword entry, U-187
- vtkFoam
 - library, U-171
- vtkPV3Foam
 - library, U-171
- vtkUnstructuredToFoam utility, U-92

W

- wall
 - boundary condition, P-63, P-69, U-59, U-139
- wall
 - keyword entry, U-139
- wallFunctionTable utility, U-91
- wallGradU utility, U-95
- wallHeatFlux utility, U-95
- Wallis

library, U-102
 wallShearStress utility, U-95
 wclean script/alias, U-76
 wdot utility, U-96
 wedge
 boundary condition, U-135, U-139, U-150
 wedge
 keyword entry, U-139
 window
 Color Legend, U-29
 Options, U-176
 Pipeline Browser, U-24, U-172
 Properties, U-173, U-174
 Render View, U-176
 Seed, U-177
 window panel
 Animations, U-176
 Annotation, U-24
 Charts, U-176
 Color Legend, U-175
 Color Scale, U-175
 Colors, U-176
 Display, U-24, U-25, U-172, U-174
 General, U-176
 Information, U-172
 Mesh Parts, U-24
 Paramters, U-173
 Properties, U-25, U-172
 Render View, U-175, U-176
 Style, U-175
 View Render, U-24
 Wireframe
 menu entry, U-175
 WM_ARCH
 environment variable, U-76
 WM_ARCH_OPTION
 environment variable, U-76
 WM_COMPILE_OPTION
 environment variable, U-76
 WM_COMPILER
 environment variable, U-76
 WM_COMPILER.BIN
 environment variable, U-76
 WM_COMPILER.DIR
 environment variable, U-76
 WM_COMPILER.LIB
 environment variable, U-76
 WM_DIR
 environment variable, U-76
 WM_MPLIB
 environment variable, U-76
 WM_OPTIONS
 OpenFOAM-3.0.0

 environment variable, U-76
 WM_PRECISION_OPTION
 environment variable, U-76
 WM_PROJECT
 environment variable, U-76
 WM_PROJECT_DIR
 environment variable, U-76
 WM_PROJECT_INST_DIR
 environment variable, U-76
 WM_PROJECT_USER_DIR
 environment variable, U-76
 WM_PROJECT_VERSION
 environment variable, U-76
 wmake
 platforms, U-73
 wmake script/alias, U-71
 word class, P-24, P-29
 writeCellCentres utility, U-96
 writeCompression keyword, U-116
 writeControl
 keyword entry, U-116
 writeControl keyword, U-22, U-62, U-116
 writeFormat keyword, U-55, U-116
 writeInterval
 keyword entry, U-181
 writeInterval keyword, U-23, U-32, U-116
 writeMeshObj utility, U-92
 writeNow
 keyword entry, U-115
 writePrecision keyword, U-116

X

x
 keyword entry, U-189
 XiFoam solver, U-89
 xmgr
 keyword entry, U-117, U-187
 xyz
 keyword entry, U-189

Y

y
 keyword entry, U-189
 yPlusLES utility, U-95
 yPlusRAS utility, U-95

Z

z
 keyword entry, U-189
 zeroGradient
 boundary condition, U-140
 zipUpMesh utility, U-94