HopeFOAM

(High Order Parallel Extensible CFD Software)

User Guide

Version 0.1

The Exercise Group
Innovation Institute for Defence Science and Technology, the
Academy of Military Science (AMS), China.

September 2017

contents

License

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

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

1. Definitions

a. **"Adaptation"** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

b. **"Collection"** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. **"Distribute"** means to make available to the public the original and copies of the Work through sale or other transfer of ownership.

d. **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

e. **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

f. **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

g. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

h. **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

i. **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

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

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

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

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

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

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested.
b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
c. If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor

institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d. For the avoidance of doubt:

    i. **Non-waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

    ii. **Waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

    iii. **Voluntary License Schemes**. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b).

e. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, 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.

About HopeFOAM

HopeFOAM is a major extension of OpenFOAM to provide higher order finite element method and other numerical methods for computational mechanics. It is developed by Exercise Group, Innovation Institute for Defense Science and Technology, the Academy of Military Science (AMS), China. The Group aims at developing open source software packages for large scale computational science and engineering. HopeFOAM has following features:

- **High Order**：In addition to the finite volume discretization method used in OpenFOAM, HopeFOAM aims at integrating high-order discretization methods into the computational mechanics Toolbox, among which DGM（Discontinuous Galerkin Method）is the first one.
- **Parallel**：In order to improve the performance and scalability of parallel computing, parallel computational toolkits/software are integrated into HopeFOAM to accelerate the discretization and computational procedures.
- **Extensible**：By incorporating with the high order discretization and efficient parallel computing, HopeFOAM provides an extensible software framework for further development of application module and easy-to-use interfaces for developers.
- **FOAM**：The current version of HopeFOAM is a major extension of the OpenFOAM-4.0 released by the OpenFOAM Foundation on the 28th of June, 2016.

HopeFOAM-0.1 is the first publicly released version of HopeFOAM and developed by a major extension of OpenFOAM-4.0. The well-known high-order discretization method, Discontinuous Galerkin Method (DGM) is implemented in HopeFOAM-0.x. There are copious references about the method in Hesthaven and Warbuton's book:【Hesthaven J S, Warburton T. Nodal discontinuous Galerkin methods: algorithms, analysis, and applications[M]. Springer Science & Business Media, 2007.】
HopeFOAM-0.1 provides 2D-DGM and related support. The major components include data structure, DGM discretization, solvers and related tools. PETSc is used for solving of linear systems of equations. 3D applications will be supported in a new release in a few months.
The guiding principle in the development of HopeFOAM-0.1 is to reuse the primitive

data structure of OpenFOAM-4.0 as much as possible and keep it consistent with the user interfaces. Thus, users of OpenFOAM could implement and adopt corresponding high order DGM solvers in a relatively straightforward way. The original OpenFOAM applications can be use normally.

This manual is the user guide of HopeFOAM-0.1, it provides a description of the basic operation of HopeFOAM through four tutorial exercises. The solver source code and command are located in the case directory, and all the cases are in the *tutorials/DG/2D* directory.

1 Compressible Isentropic Vortex

This tutorial describes how to pre-process, run, post-process a case involving inviscid compressible isentropic vortex in a two-dimensional square domain. The geometry is shown in Figure 1.1: $0\text{m} \le x \le 10\text{m}$, $-5\text{m} \le y \le 5\text{m}$ in 2D x-y coordinates.

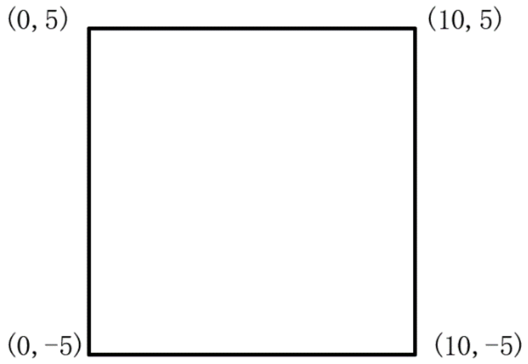


Figure 1.1 Geometry of the isentropic vortex

The analytical solutions of compressible isentropic vortex are

$$u = u_0 - \beta e^{(1-r^2)} \frac{y - y_0}{2\pi},$$
$$v = v_0 + \beta e^{(1-r^2)} \frac{x - x_0 - t}{2\pi},$$
$$\rho = \left(1 - \left(\frac{\gamma - 1}{16\gamma\pi^2}\right)\beta^2 e^{2(1-r^2)}\right)^{\frac{1}{\gamma-1}},$$
$$p = \rho^\gamma,$$

where $r = \sqrt{(x - t - x_0)^2 + (y - y_0)^2}$, $x_0 = 5$, $y_0 = 0$, $u_0 = 1$, $v_0 = 0$, $\beta = 5$, $\gamma = 1.4$, which indicates that the vortex moves from the center of the domain to the right at a speed of 1m/s.

1.1 Mesh generation

To run the isentropicVortex case, please change to the case directory firstly.

```
cd $FOAM_RUN/tutorials/DG/2D/isentropicVortex
```

The isentropicVortex domain is a unit square in the xy plane, all the boundaries are **Wall** types. Both structure and unstructured mesh format are supported by HopeFOAM. The mesh tool supplied by OpenFOAM, blockMesh, could be used directly to generate structure mesh for HopeFOAM by configuring *system/blockMeshDict*. The *blockMeshDict* entries for this case are shown below:

```
boundary
(
   Wall
   {
      type wall;
      faces
      (
         (3 7 6 2)
         (0 4 7 3)
         (2 6 5 1)
         (1 5 4 0)
      );
   }
   frontAndBackPlanes
   {
      type empty;
      faces
      (
         (0 3 2 1)
         (4 5 6 7)
      );
   }
);
```

To generate the mesh, run the blockMesh utility by typing in the terminal:

```
blockMesh
```

Alternatively, mesh can also be imported from third-party mesh utilities, such as gambit, pointwise, fluent ICEM et al. In this case, we provide three unstructure fluent mesh files with different mesh scale: *vortex0256.msh*, *vortex1024.msh* and *vortex4096.msh*. Their corresponding characteristic lengths are h, $h/2$ and $h/4$ as shown in Figure 1.2. Mesh can be generated by typing in the terminal:

```
fluentMeshToFoam vortex0256.msh
```

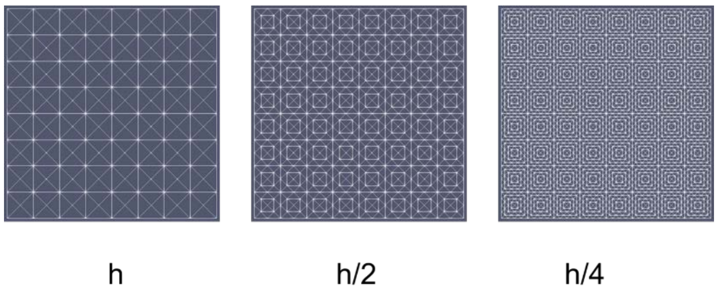


Figure 1.2 Meshes with different characteristic lengths

Users can refer to Chapter 5 of *OpenFOAM-User-Guide* for more information about mesh generation.

1.2 Boundary and initial conditions

Simple boundary conditions (fixedValue, zeroGradient) can be imposed by configure patch types in field file for HopeFOAM. However, boundary of

IsentropicVortex is described by complex formulations standing for analytical solutions. Herein, the analytical solutions are coded in *setNonUniformInlet.H* and *setBoundaryValues.H* files, located in dgEulerFoam solver directory. The governing equations can be written as follows:

$$\frac{\partial \boldsymbol{q}}{\partial t}+\frac{\partial \boldsymbol{F}}{\partial x}+\frac{\partial \boldsymbol{G}}{\partial y}=0$$

$$\boldsymbol{q}=\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ E\end{pmatrix},\quad \boldsymbol{F}=\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ u(E+p)\end{pmatrix},\quad \boldsymbol{G}=\begin{pmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ v(E+p)\end{pmatrix}$$

The initial fields are stored in the ***0*** sub-directory. File *rho* sets initial values and boundary conditions of density ρ , file *rhoU* corresponds to the momentum vector $(\rho u,\rho v)$, file Ener corresponds to the energy E , file *U* corresponds to the velocity vector (u,v) , file *p* corresponds to the pressure p . *rho*, *rhoU* and Ener are primitive variables, while *U* and *p* are derived by formulation:

$$E=\frac{p}{\gamma-1}+\frac{\rho}{2}(u^2+v^2)\,.$$

Besides the value of boundary and internal fields, which are set by analytical solution, the patches type need to be specified. For scalar fields *rho*, *p*, *Ener*, `internalField` and `boundaryField` are set as follows:

```
internalField   uniform 0;

boundaryField
{
  Wall
   {
       type          fixedValue;
       value         uniform 0;
   }

   frontAndBackPlanes
   {
       type          empty;
   }
}

//************************************************************** //
```

While the entries of vector fields *U*, *rhoU* for `internalField` and `boundaryField` are set as follows:

```
internalField   uniform (0 0 0);

boundaryField
{
   Wall
   {
      type          fixedValue;
      value         uniform (0 0 0);
   }

   frontAndBackPlanes
      {
         type          empty;
      }
}

//*********************************************************************** //
```

It should be noted that, the useful information above is patch type entries, while values are meaningless.

1.3 Physical properties

Dimensionless gas constant γ must be specified for isentropicVortex case. It could be configured in the file *transportProperties* under the *constant* sub-directory. The keyword for gas constant is `gamma`, and below is an example for the dictionary:

```
   gamma      gamma [ 0 0 0 0 0 0 0 ] 1.4;

// ***************************************************** //
```

1.4 Discretization and solver settings

Discontinuous galerkin(DG) method discretization, flux and limiter schemes are specified in the file *dgSchemes* under *system* directory. Compared with the OpenFOAM-4.0, HopeFOAM-0.1 keeps the default parameter settings of discretization schemes. Entry `godunovScheme` is designed for compressible NS simulation, controlling flux and limiter scheme.

```
ddtSchemes
{
   default           Euler;
}

gradSchemes
{
   default           default none;
   grad(p)           default none;
   grad(gther_p)      default none;
}

godunovScheme
{
   fluxScheme         Roe;
   limiteScheme       Triangle;
}
```

Specification of the DG polynomial order, solver tolerances and other algorithm configurations are stored in the *dgSolution* dictionary. The *dgSolution* dictionary offers a sub-dictionary “`DG`”, which contains the configurations of dimension and order. In current version, `baseOrder` can be set ranging from 1 to 8.

```
DG
{
   meshDimension      2;
   baseOrder         4;
}

solvers
{
   dgrho
   {
      tolerance      1e-12;
      relTol        0;
      kspSolver      preonly;
   }

   rhoU
   {
      tolerance      1e-12;
      relTol        0;
      kspSolver      preonly;
   }

   Ener
   {
      tolerance      1e-12;
      relTol        0;
      kspSolver      preonly;
   }

   "rho(1|2|3)"
   {
      tolerance      1e-12;
      relTol        0;
      kspSolver      preonly;
   }

   "rhoU(1|2|3)"
   {
      tolerance      1e-12;
      relTol        0;
      kspSolver      preonly;
   }

   "Ener(1|2|3)"
   {
      tolerance      1e-12;
      relTol        0;
      kspSolver      preonly;
   }
}

PISO
{
   nCorrectors    2;
   nNonOrthogonalCorrectors 0;
   pRefCell       0;
   pRefValue      0;
}

// ****************************************************************** //
```

1.5 Control

The information relating to the control of the solution procedure, including time step，start time and end time, is stored in the *system/controlDict* dictionary. For this case, the `startTime` is set to `0`. To guarantee the vortex not pass through this domain, end time should be less than 5s. The `writeInterval` is set to `25`. Time step `deltaT` is decided by the mesh size and the order approximation to satisfy the CFL restriction (the calculation method refer to *Nodal Discontinuous Galerkin Methods Algorithms Analysis and Applications* Section 6.4). The *controlDict* entries are written as follows:

```
application     dgEulerFoam;

startFrom        startTime;

startTime        0;

stopAt          endTime;

endTime         2;

deltaT          0.008;

writeControl    timeStep;

writeInterval    25;

purgeWrite       0;

writeFormat      ascii;

writePrecision    16;

writeCompression  off;

timeFormat        general;

timePrecision     6;

runTimeModifiable true;

adjustTimeStep    no;

maxCo             1;

maxDeltaT         1;

// ******************************************************************** //
```

A stable `deltaT` configuration for various base order and mesh size is given in Table 1.1.

Table1.1 `DeltaT` of different meshes and different order approximations

baseOrder	vortex0256	vortex1024	vortex4096
1	0.04	0.02	0.01
2	0.02	0.01	0.005
3	0.008	0.004	0.002
4	0.008	0.004	0.002
5	0.004	0.002	0.001
6	0.002	0.001	0.0005

1.6 Running the code

Before running the code, please use wmake under the source code directory to generate executable solver. The solver is executed by entering the case directory and typing:

```
    $ ./dgEulerFoamVortex
```

1.7 Post-processing

vortex1024.msh is used and `baseOrder` is set to 4. After computational results are written to time directories, the *VTK* folder could be generated by typing in the terminal:

```
    $ dgToVTK
```

, which is a high order post-processing utility released within HopeFOAM. The results can be viewed by ParaView， clicking `file>Open...` to choose .vtk files, as shown in Figure1.3.

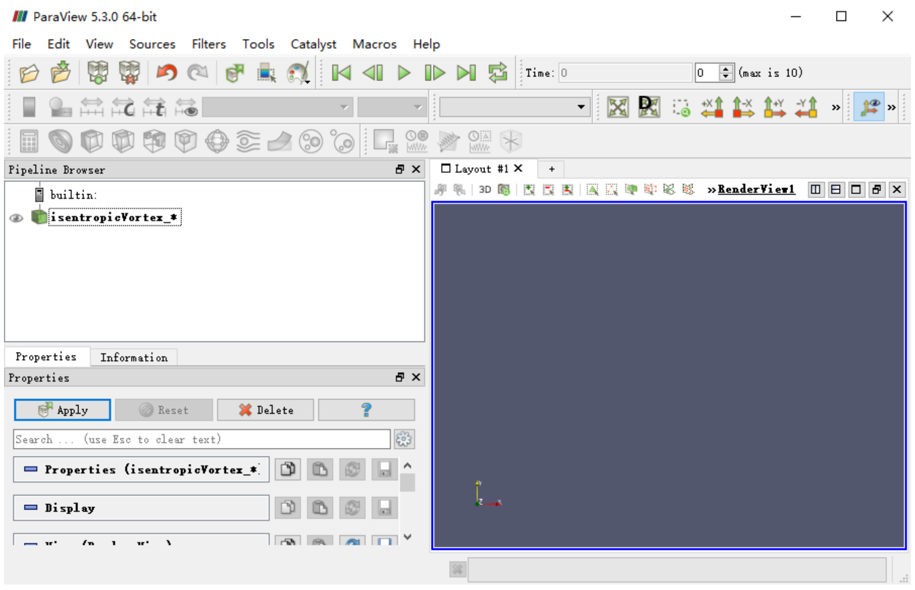


Figure1.3 ParaView window

To view contour plots as shown in Figure 1.4, click `apply` button and select rho, rhoU, Ener from the Coloring menu.

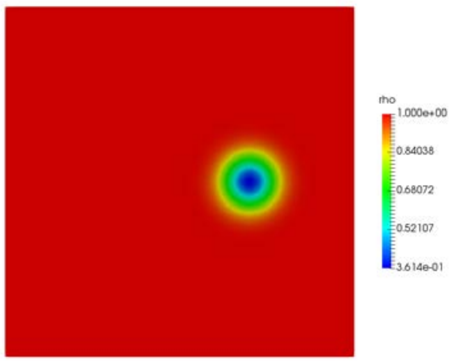


Figure1.4 Density in the isentropicVortex case

Please refer to Chapter 2 of OpenFOAM-User-Guide for more information about Post-processing.

1.8 Computing convergence rate

At the end of the simulation, the errors in L2 norm of numerical solution are calculated by *dgEulerFoam /eulererror.H.*

```
rhoError: 8.807979526797244e-06
rhoUError: 1.865574862711117e-05
```

The rate of convergence can be calculated with a sequence of nested refined meshes, as defined by:

$$\log_2\left(\frac{\varepsilon_h}{\varepsilon_{h/2}}\right)$$

where ε_h is the density or momentum error of the mesh with characteristic length h.

1.9 Running in parallel

For large scale problems, parallel processing supported by HopeFOAM could significantly reduce the computational time.

The configuration dictionary for parallel processing named *decomposeParDict* locates in the *system* directory. The first entry is `numberOfSubdomains`, which specifies the number of decomposed subdomains. Specifically, numberOf-Subdomains should be consistent to the number of processors used to run the solver. Within the simpleCoeffs sub-dictionary, the vector n specifies the number of subdomains in x, y and z directions. The number of subdomains specified by $n_x * n_y * n_z$ should equal to numberOfSubdomains. If the geometry is two-dimensional, we should set $n_z = 1$.

```
numberOfSubdomains 12;

method simple;

simpleCoeffs
{
   n       (3 4 1);
   delta    0.001;
}
// ************************************************************************* //
```

After running dgDecomposePar, the geometry and associated fields are decomposed into 12 pieces. The case directory generates 12 folders from *processor0* to *processor11.* Parallel simulation will start by typing:

```
$ mpirun -np 12 ./dgEulerFoamVortex –parallel
```

Once the case has finished running, the decomposed fields and mesh are reassembled for post-processing using the dgReconstructPar utility. Simply execute it from the command line:

```
$ dgReconstructPar
```

2 Double Mach Reflection

Double Mach reflection uses the same solver as isentropic vortex. For the double Mach reflection case simulation, we modify the *setNonUniformInlet.H*, *setBoundaryValues.H*, and *eulererror.H* files. Notice that different from solver in IsentropicVortex case, comment annotation of line 91 and 118 in *dgEulerFoam/dgEulerFoam.C* are removed to active the limiter. For problems with shocks, limiter is required to stabilize the simulation.

```
 91         Godunov.limite(rho1,rhoU1,Ener1);
118         Godunov.limite(rho,rhoU,Ener);
```

Recompile the solver by typing the following command in terminal window:

```
    $ wclean
    $ wmake
```

The source code of the modified `dgEulerFoam` solver and `dgEulerFoamDouble` command are located in the *tutorials/DG/2D /doubleMach* case directory.

This case involves a Mach 10 shock in the air (γ=1.4) with an initial angle of 60° to the reflecting wall. The schematic diagram is shown in Figure 2.1.

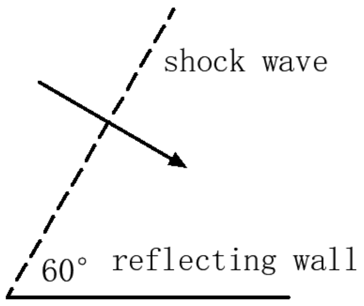


Figure 2.1 Double Mach reflection schematic diagram

The two-dimensional domain is shown in Figure 2.2 with $0 \le x \le 3.2$ and $0 \le y \le 1$. A Mach 10 shock is set up with an initial position of x=0.16667 on the lower boundary. The shock makes an angle of 60° to the x axis and extends to the top of the domain at $y = 1$. Special fixed boundary conditions are set up for `inlet` and `wall` patches at $y = 0$, and $0 \le x \le 0.16667$ allowing the shock to propagate off the domain. The reflecting wall named `wall` patch lies along the bottom of the problem domain, beginning at x=0.16667. For the upper boundary $y = 1$ a time-dependent boundary is imposed to allow the shock to propagate onto the domain as though it extended to infinity. The values along the top boundary including `inlet` and `far` patches are set to describe the exact motion of the initial Mach 10 shock. The simulation is run until $t = 2$, by which time the shocks should be nearly at the `outlet` patch. The undisturbed air ahead of the shock has a density of 1.4 and a pressure of 1.

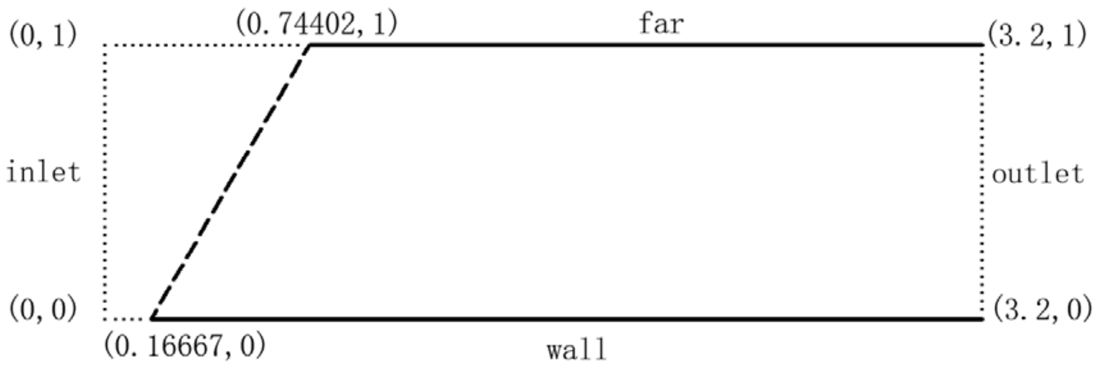


Figure 2.2 Geometry and boundary of Double Mach reflection

2.1 Mesh Generation

We provide a *doubleMach.msh* mesh file under doubleMach case directory, the mesh is generated by typing in the terminal:

```
$ fluentMeshToFoam doubleMach.msh
```

2.2 Boundary and initial conditions

Similar with Section 1.2, the boundary and initial conditions are programmed into dgEulerFoam solver, we add *setNonUniformInlet.H* and *setBoundaryValues.H* files to dgEulerFoam solver source code. The entries for the scalar files *rho*, *p*, *Ener* in 0 sub-directory are configured as follows:

```
internalField   uniform 0;

boundaryField
{
   wall
   {
      type          reflective;
      value         uniform 0;
   }

   far
   {
      type          fixedValue;
      value         uniform 0;
```

```
    }

    inlet
    {
        type            fixedValue;
        value           uniform 0;
    }

    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }

    frontAndBackPlanes
    {
        type            empty;
    }
}

//******************************************************************* //
```

The entries for the vector fields *U* and *rhoU* are shown as belowSSA zx:

```
internalField   uniform (0 0 0);

boundaryField
{
    wall
    {
        type            reflective;
        value           uniform (0 0 0);
    }

    far
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    inlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    outlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    frontAndBackPlanes
    {
        type            empty;
    }
}

//******************************************************************* //
```

Note that the `wall` patch are given a `reflective` boundary condition.

2.3 Running the code

Before running the code, please use wmake under the source code directory to generate executable solver. Configurations of *constant/transportProperties* and *system/dgSchemes* dictionaries are set the same as isentropicVortex case. Ensure that the `baseOrder` in the dgSolution dictionary is set to `1`. In the *system/controlDict* directory, set `deltaT` to 1e-5 and `endTime` to `0.2`.

This case typically takes a few hours when running in serial. To decrease the running time, edit *system/decomposeParDict* as referred to section 1.9. Then decompose the mesh and fields:

```
$ dgDecomposePar
```

Then run in parallel:

```
$ mpirun -np 12 ./dgEulerFoamDouble –parallel
```

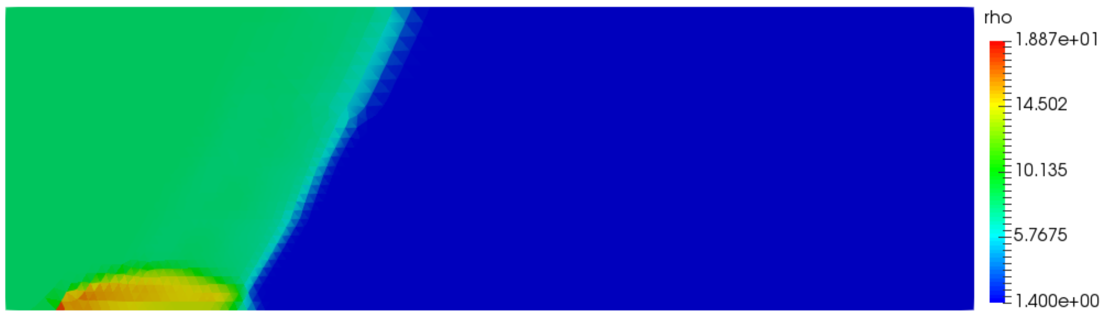
At last, reassemble the fields and mesh data:

```
$ dgReconstructPar
```

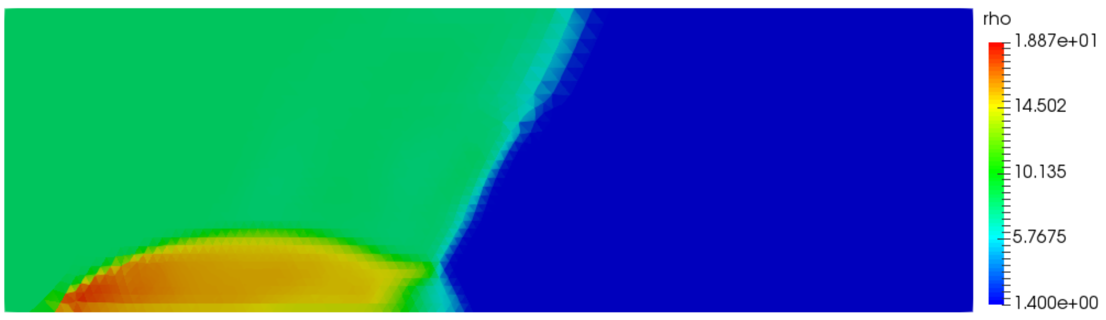
2.4 Post-processing

Use the commend `dgToVTK` to generate the *VTK* folder. Then open the ParaView software and click the button `file>Open...` to choose .vtk files. To view density contour plots as shown in Figure 2.3, please open the Properties panel, select rho from the Coloring menu. Then click `play` in VCR controls panel to view density at different times.

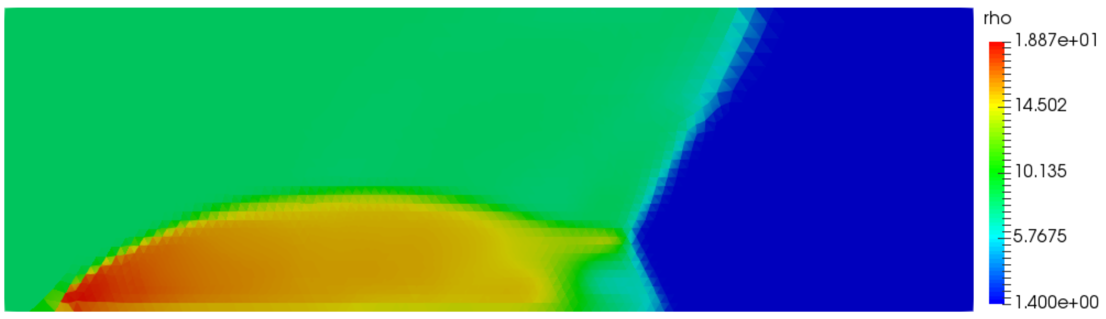
The limiter in HopeFOAM-0.1 is first order and does not support higher order calculations. We provide encryption mesh file *doubleMachDense.msh* for fine flow field.



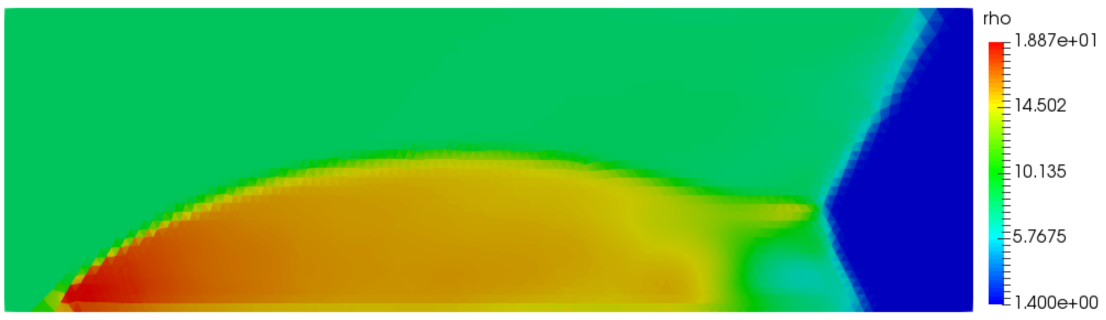
$t = 0.05$



$t = 0.1$



$t = 0.15$



$t = 0.2$

Figure 2.3 Density in the doubleMach case

3 Incompressible vortex

This tutorial describes how to pre-process, run, post-process a case involving incompressible vortex in a two-dimensional square domain. Two-dimensional Navier-Stokes equations for viscous laminar flow are implemented into a HopeFOAM solver named dgChorinFoam.

In this particular case there are four inflow and four outflow regions, as shown in Figure 3.1: $-0.5\text{m} \le x \le 0.5\text{m}$, $-0.5\text{m} \le y \le 0.5\text{m}$.

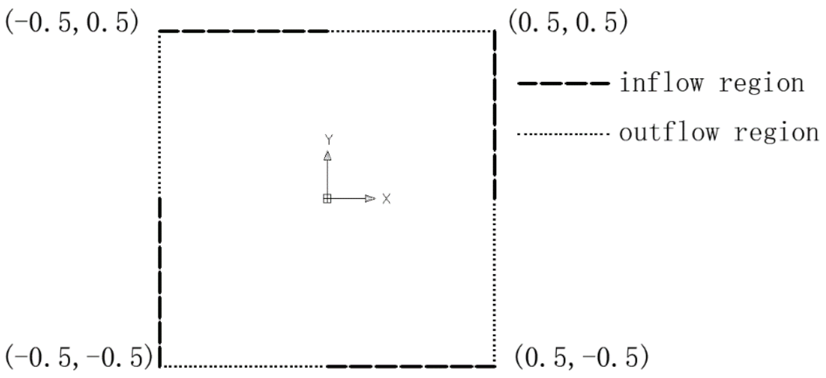


Figure 3.1 Geometry of the incompressible vortex

3.1 Mesh generation

After changing to the pearsonVortex case directory, the mesh divides to 4 blocks, the block structure is shown in Figure 3.2.

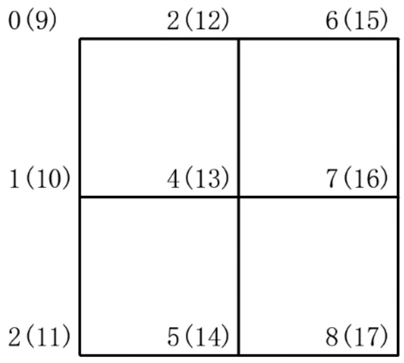


Figure 3.2 Block structure for incompressible vortex

The *blockMeshDict* entries for this case are as follows:

```
convertToMeters 1;

vertices
(
    (-0.5 0.5 0)
    (-0.5 0 0)
```

```
   (-0.5 -0.5 0)
   (0 0.5 0)
   (0 0 0)
   (0 -0.5 0)
   (0.5 0.5 0)
   (0.5 0 0)
   (0.5 -0.5 0)
   (-0.5 0.5 0.1)
   (-0.5 0 0.1)
   (-0.5 -0.5 0.1)
   (0 0.5 0.1)
   (0 0 0.1)
   (0 -0.5 0.1)
   (0.5 0.5 0.1)
   (0.5 0 0.1)
   (0.5 -0.5 0.1)
);

blocks
(
   hex (0 1 4 3 9 10 13 12) (8 8 1) simpleGrading (1 1 1)
   hex (1 2 5 4 10 11 14 13) (8 8 1) simpleGrading (1 1 1)
   hex (3 4 7 6 12 13 16 15) (8 8 1) simpleGrading (1 1 1)
   hex (4 5 8 7 13 14 17 16) (8 8 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
   inlet
   {
      type wall;
      faces
      (
         (2 11 10 1)
         (8 17 14 5)
         (6 15 16 7)
         (0 9 12 3)
      );
   }
   outlet
   {
      type wall;
      faces
      (
         (1 10 9 0)
         (5 14 11 2)
         (7 16 17 8)
         (3 12 15 6)
      );
   }
   frontAndBackPlanes
   {
      type empty;
      faces
      (
         (1 0 3 4)
         (10 13 12 9)
         (2 1 4 5)
         (11 14 13 10)
         (5 4 7 8)
         (14 17 16 13)
         (4 3 6 7)
         (13 16 15 12)
      );
```

```
    }
);

mergePatchPairs
(
);

// ************************************************************************* //
```

Mesh is generated by typing in the terminal:

```
$ blockMesh
```

3.2 Boundary and initial conditions

Similar with Section 1.2, pearsonVortex case uses analytical solution to set initial values and boundary conditions, which is implemented by adding *setNonUniformInlet.H* and *setBoundaryValues.H* files to dgChorinFoam solver source code.

$$u = -\sin(2\pi y)e^{-\nu 4\pi t},$$
$$v = \sin(2\pi x)e^{-\nu 4\pi t},$$
$$p = -\cos(2\pi x)\cos(2\pi y)e^{-\nu 8\pi t}.$$

Since the boundary and initial conditions are programmed into dgChorinFoam solver, the field value in the *0/U* file is meaningless. Here, `internalField` and `inlet` patch are simply initialized as uniform zero, which is expressed by 3 vector components `(0 0 0)`. Neumann boundary condition for the velocity at the outflow boundary is assumed by a `fixedGradient` condition with a value of uniform `(1 1 0)`.

```
dimensions     [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type           fixedValue;
        value          uniform (0 0 0);
    }

    outlet
    {
        type           fixedGradient;
        gradient        uniform (1 1 0);
    }

    frontAndBackPlanes
    {
        type           empty;
    }
}
```

```

// ************************************************************************* //
```

Similarly, we set the pressure field in the `0/p` file. Neumann boundary condition for the pressure on the `inlet` patch are set as `zeroGradient`. At the `outlet` patch, a zero pressure is assumed.

3.3 Physical properties

The physical properties for the pearsonVortex case is stored in the *constant/ transportProperties* dictionary. The keyword for kinematic viscosity is `nu`. The keyword `transportModel` represents fluid type and is set as `Newtonian`. The *transportProperties* entries for this case are as follows:

```
transportModel  Newtonian;
nu            nu [ 0 2 -1 0 0 0 0 ] 0.01;

// ************************************************************************* //
```

3.4 Discretisation and solver settings

Keep the default configurations for the discretization schemes in *dgSchemes* dictionary under *system* directory. The *dgSolution* dictionary contains a sub-dictionary `DG,` which specifies dimension and order approximation. The dimension of the mesh `meshDimension` is set to `2`. The order approximation `baseOrder` can be set ranging from 1 to 8. The `solver` tolerance should be set as 10^{-12} to ensure high order convergence rate.

3.5 Running the code

Before running the code, please use wmake under the source code directory to generate executable solver. To stabilize the computation, we set `deltaT` to 0.001 and `endTime` to `0.1` in *ControlDict* dictionary, while `baseorder` is `2`. Run the case by typing in the terminal:

```
$ ./dgChorinFoamVortex
```

3.6 Post-processing

After generating *VTK* files by the command `dgToVTK`, we use paraView to plot the vectors of the flow velocity.

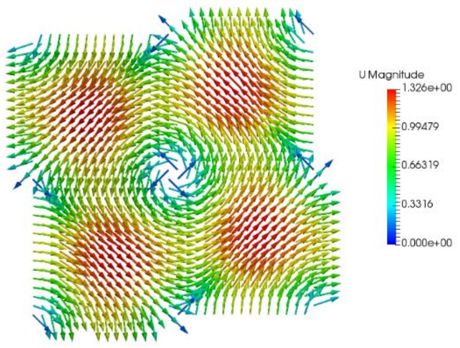


Figure 3.3 Velocities in the pearsonVortex case

3.7 Computing rate of convergence

Similar to the isentropicVortex example, we give three sets of meshes and their reference values for `deltaT` at different order approximation to calculate the rate of convergence, as shown in Table 3.1. `DeltaT` is proportional to characteristic length h. At the end of the simulation, the errors of numerical solution are outputed .

Table3.1 `DeltaT` of different meshes and different order approximations

baseOrder	h	0.5h	0.25h
1	0.002500000	0.001250000	0.000625000
2	0.001100000	0.000555560	0.000285710
3	0.000625000	0.000322580	0.000161290
4	0.000400000	0.000204080	0.000103090
5	0.000285710	0.000142860	0.000071429
6	0.000208330	0.000105260	0.000052632

4 Unsteady flow around cylinder

Unsteady flow around cylinder in channel uses the same solver as the incompressible vortex case. To simulate this case, we modify the *setNonUniformInlet.H*, *setparaT.H*, *pEqnCorrect.H* and *setBoundaryValues.H* files. This case requires to modify the *dgEulerFoam/dgEulerFoam.C* file as follow:

- Add *createTimeControls.H* file

```
45    #include "createTimeControls.H"
```

- Set parameters at each time step

```
70    paraT4 = std::sin(pi*runTime.value()/8)/paraT3;
71    paraT3 = std::sin(pi*runTime.value()/8);
```

- `paraT3` modified as `paraT1`

```
86    shared_ptr<dg::Equation<scalar>>    result1    =
       make_shared<bCorrectEquation<scalar>>(dgm::lap
       lacian(p), paraT1);
```

- Add line 116 annotation comment `//`:

```
116   //  #include "chorinerror.H"
```

Recompile the solver and generated binary file `dgChorinFoamCylinder` is located in the *cylinder* case directory.

This case simulates a time-dependent two-dimensional flow through a channel, with a cylinder placed off the centerline of the channel. It begins to shed vortices as soon as the parabolic inflow is ramped up from its zero initial condition. The domain is shown in Figure 4.1, the center of cylinder locates at origin $(0,0)$ and its radius is 0.05.

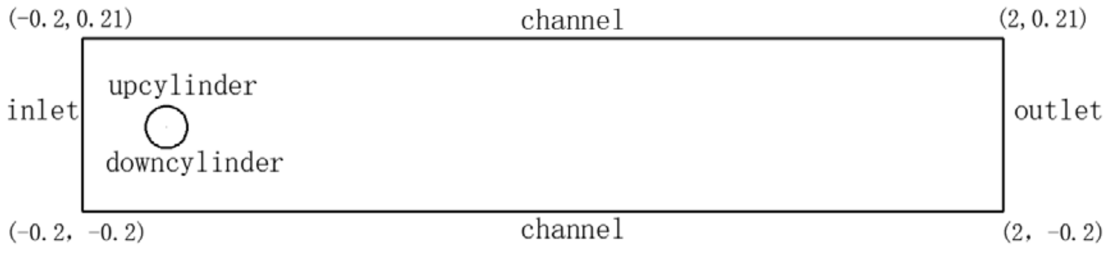


Figure 4.1 Geometry and boundary of unsteady flow around cylinder

At initial time $t=0$, zero flow is assumed. The simulations ends at $t=8$. The inflow boundary conditions for this case are given by a parabolic inflow profile, modulated by a sine function depending on time as:

$$u(x,y,t) = 0.41^{-2}\sin(\frac{\pi t}{8})6(y+0.2)(0.21-y),$$
$$v(x,y,t) = 0.$$

4.1 High order curved boundary

Class `arcPolyPatch` is introduced in HopeFOAM for supporting high-order representation of curved boundary. Detailed configuration locates in the *constant/polyMesh/boundary* dictionary. We take `upcylinder` as an example to illustrate how to configure curved boundary. Boundary type entry `type` is set to `arc` and `name` specifies the surface boundary name. The parametric equation of the 2D curved boundary is marked with notation `#{ #}`, which describes the relationship between x, y, z coordinates and parameter `u`, `v.` `u_Range` specifies that the value of parameter `u` is in the range of [-0.5 0.5].

```
6
(
    channel
    {
        type            wall;
        inGroups         1(wall);
        nFaces          128;
        startFace        2666;
    }
    outlet
    {
        type            patch;
        inGroups         1(wall);
        nFaces          12;
        startFace        2794;
    }
    inlet
    {
        type            patch;
        inGroups        1(wall);
        nFaces         16;
        startFace       2806;
    }
    upcylinder
    {
        type            arc;
        inGroups         1(wall);
        nFaces          16;
        startFace        2822;
        name            codeup;
        u_Range          (-0.5 0.5);
        v_Range          (0 0);
        code
        #{
               0.05*Foam::sin(Foam::constant::mathematical::pi*u),
               0.05*Foam::cos(Foam::constant::mathematical::pi*u),
               v
        #};
    }
    downcylinder
    {
        type          arc;
        inGroups       1(wall);
        nFaces        16;
        startFace      2838;
        name          codedown;
        u_Range        (-0.5 0.5);
        v_Range        (0 0);
        code
        #{
               0.05*Foam::sin(Foam::constant::mathematical::pi*u),
              -0.05*Foam::cos(Foam::constant::mathematical::pi*u),
               v
        #};
    }

    frontAndBackPlanes
    {
        type          empty;
        inGroups       1(empty);
        nFaces        3680;
        startFace      2854;
    }
)

//******************************************************************* //
```

4.2 Boundary and initial conditions

Similar with Section 1.2, velocity field and pressure field of `inlet` patch are set by formulas. Natural boundary conditions for the velocity on the `outlet` patch is set to `fixedGradient`. No slip boundary conditions are imposed on the `upcylinder`, `downcylinder` and `channel` walls by setting `fixedValue` type with `value` of `uniform (0 0 0)`.

```
boundaryField
{
    inlet
    {
       type          fixedValue;
       value          uniform (0 0 0);
    }

    outlet
    {
       type          zeroGradient;
    }

    channel
    {
       type          fixedValue;
       value          uniform (0 0 0);
    }

    upcylinder
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    downcylinder
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    frontAndBackPlanes
    {
        type            empty;
    }
}

//******************************************************************* //
```

The pressure of the boundary `upcylinder`, `downcylinder` and `channel` are set as `zeroGradient` condition. The `outlet` patch is assumed as zero pressure boundary.

```
boundaryField
{
    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }

    channel
    {
        type            zeroGradient;
    }

    upcylinder
    {
        type            zeroGradient;
    }

    downcylinder
    {
        type            zeroGradient;
    }

    frontAndBackPlanes
    {
        type            empty;
    }
}

//******************************************************************* //
```

4.3 Physical properties

This case runs with a Reynolds number of 100. The kinematic viscosity `nu` is configured by the *constant/transportProperties* dictionary with a value of 0.001. Order approximation `baseOrder` is set to `3` in *system/dgSolution* dictionary.

There is no closed-form analytical solution for this problem. Instead, we compute the drag and lift coefficients for the cylinder, which are defined by:

$$C_d(t) = -\oint_{Cylinder} -p\hat{n}_x + \nu(\hat{n}_x 2\frac{\partial u}{\partial x} + \hat{n}_y(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}))\,\mathrm{d}\,s,$$

$$C_l(t) = -\oint_{Cylinder} -p\hat{n}_y + \nu(\hat{n}_y 2\frac{\partial v}{\partial y} + \hat{n}_x(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}))\,\mathrm{d}\,s,$$

where $(\hat{n}_x, \hat{n}_y)$ is the outward normal along the cylinder surface.

In the *controlDict* directory, the `forceCoeffs` is specified in the `functions` sub-dictionary and `deltaT` is set to `2E-4`. The *controlDict* entries are presented as follows:

```
application     dgChorinFoam;

startFrom        startTime;

startTime       0;

stopAt           endTime;

endTime          8;

deltaT           2.0E-04;

writeControl     timeStep;

writeInterval    200;

purgeWrite        0;

writeFormat       ascii;

writePrecision   6;

writeCompression off;

timeFormat        general;

timePrecision    6;

runTimeModifiable false;

adjustTimeStep   no;

maxCo             0.9;

maxDeltaT        0.01;

 functions
```

```
{
  #include "forceCoeffs"
};

//******************************************************************* //
```

The calculation of drag and lift coefficients is specified in system/forceCoeffs dictionary. `rhoInf` refers to the reference density and `magUInf` specifies reference velocity. Both of them are set to `1` for convenience. `liftDir` and `dragDir` respectively indicate the direction of lift and drag force.

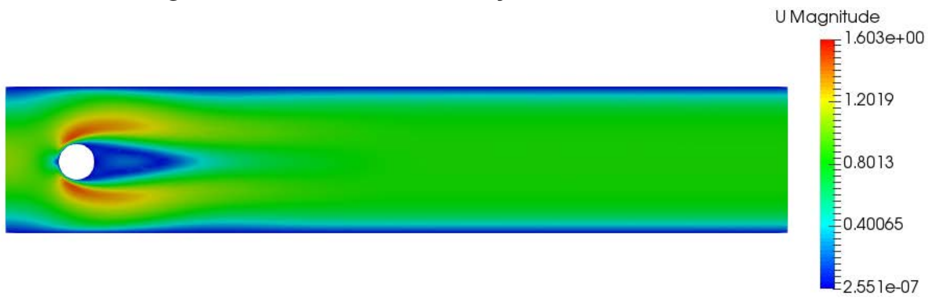
```
forceCoeffs
{
    type dgForceCoeffs;
    functionObjectLibs ("libdgforces.so");
    patches (upcylinder downcylinder);
    log true;
    pName p;
    Uname U;
    rho rhoInf;
    rhoInf 1;
    magUInf 1;
    liftDir (0 1 0);
    dragDir (1 0 0);
    pitchAxis (0 0 -1);
    CofR (0 0 0);
    Aref 0.1; //2D example default height of the grid is 1
    lRef 1;
    outputControl  timeStep;
    outputInterval 1;
}

// ******************************************************************* //
```

4.4 Running the code and post-processing

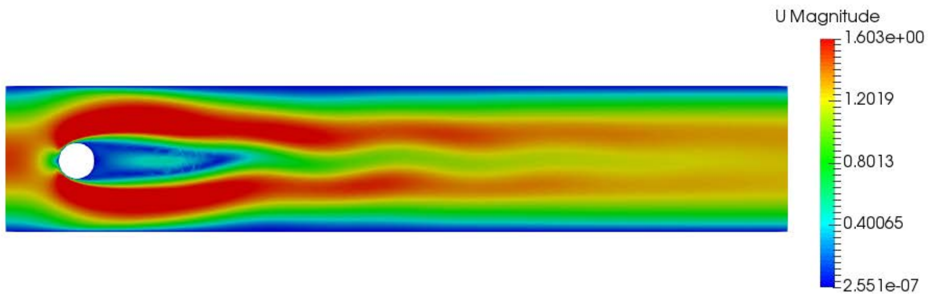
Before running the code, please use wmake under the source code directory to generate executable solver. Currently, HopeFOAM-0.1 does not support parallel computing with curved boundaries. Run the code by typing in the terminal:

```
$ ./dgChorinFoamCylinder
```

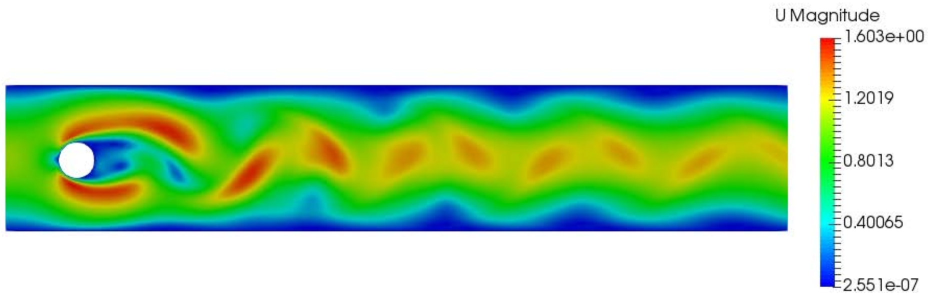
Use paraView to plot nephogram of the flow velocity. Coefficients are stored in .dat file in *postProcessing/forceCoeffs/0/directory.*



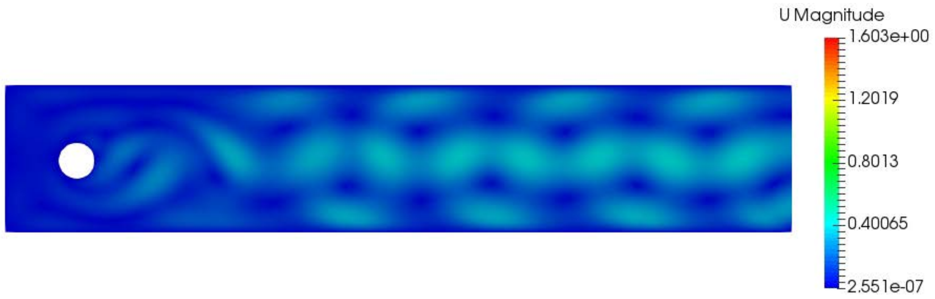
$t = 2$



$t = 4$



$t = 6$



$t = 8$

Figure 4.2 Velocity in cylinder case