Compressible Flow Solver: Navier Stokes equations

Tutorials

May 15, 2024

Department of Aeronautics, Imperial College London, UK
Scientific Computing and Imaging Institute, University of Utah, USA

2

Chapter 1

Introduction

The aim of this tutorial is to introduce the user to the spectral/*hp* element framework *Nektar++* and to describe the main features of its Compressible Flow Solver in a simple manner. If you have not already downloaded and installed *Nektar++*, please do so by visiting www.nektar.info, where you can also find the User-Guide with the instructions to install the library.

This tutorial requires:

- *Nektar++* CompressibleFlowSolver and pre- and post-processing tools,
- The visualisation tool Paraview or VisIt

1.1 Goals

After the completion of this tutorial, you will be familiar with:

- The setup of the initial and boundary conditions, the parameters and the solver settings;
- The expansions set up to mitigate aliasing effects;
- The addition of artificial viscosity to deal with flow discontinuities and the consequential numerical oscillations;
- Running a simulation with the CompressibleFlow solver;
- The post-processing of the data and the visualisation of the results in Paraview or VisIt;
- The creation of Paraview animation to monitor the evolution of the simulation or visualize non-steady simulations; and
- The use of FieldConvert modules to extract useful quantities from the field variables.

Task 1.1

Prepare for the tutorial. Make sure that you have:

- Installed and tested *Nektar++* v5.6.0 from a binary package, or compiled it from source. By default binary packages will install all executables in `/usr/bin`. If you compile from source they will be in the sub-directory `dist/bin` of the `build` directory you created in the *Nektar++* source tree. We will refer to the directory containing the executables as `$NEK` for the remainder of the tutorial.
- Downloaded the tutorial files: http://doc.nektar.info/tutorials/5.6.0/cfs/CylinderSubsonic_NS/cfs-CylinderSubsonic_NS.tar.gz Unpack it using `unzip cfs-CylinderSubsonic_NS.tar.gz` to produce a directory `cfs-CylinderSubsonic_NS` with subdirectories called `tutorial` and `complete` We will refer to the `tutorial` directory as `$NEKTUTORIAL`.

Task 1.2

Additionally, you should also install

- a visualization package capable of reading VTK files, such as ParaView (which can be downloaded from here) or VisIt (downloaded from here). Alternatively, you can generate Tecplot formatted .dat files for use with Tecplot.

1.2 Background

The Compressible Flow Solver allows us to solve the unsteady compressible Euler and Navier-Stokes equations for 1D/2D/3D problems using a discontinuous representation of the variables. For a more detailed description of this solver, please refer to the User-Guide.

In this tutorial we focus on the 2D Compressible Navier-Stokes equations. The two-dimensional second order partial differential equations can be written as:

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} + \frac{\partial \mathbf{g}}{\partial y} = 0, \tag{1.1}$$

where $\mathbf{q}$ is the vector of the conserved variables,

$$\mathbf{q} = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{Bmatrix} \tag{1.2}$$

where ρ is the density, u and v are the velocity components in x and y directions, p is the pressure and E is the total energy. In this work we considered a perfect gas law for which the pressure is related to the total energy by the following expression:

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho(u^2 + v^2), \tag{1.3}$$

where γ is the ratio of specific heats.

The vector of the fluxes $\mathbf{f} = \mathbf{f}(\mathbf{q}, \nabla(\mathbf{q}))$ and $\mathbf{g} = \mathbf{g}(\mathbf{q}, \nabla(\mathbf{q}))$ can also be written as:

$$\mathbf{f} = \mathbf{f}_i - \mathbf{f}_v, \quad \mathbf{g} = \mathbf{g}_i - \mathbf{g}_v, \tag{1.4}$$

The inviscid fluxes $\mathbf{f}_i$ and $\mathbf{g}_i$ take the form:

$$\mathbf{f}_i = \begin{Bmatrix} \rho u \\ p + \rho u^2 \\ \rho u v \\ u(E+p) \end{Bmatrix}, \quad \mathbf{g}_i = \begin{Bmatrix} \rho v \\ \rho u v \\ p + \rho v^2 \\ v(E+p) \end{Bmatrix}, \tag{1.5}$$

while the viscous fluxes $\mathbf{f}_v$ and $\mathbf{g}_v$ take the following form:

$$\mathbf{f}_v = \begin{Bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ u\tau_{xx} + v\tau_{yx} + kT_x \end{Bmatrix}, \quad \mathbf{g}_v = \begin{Bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ u\tau_{xy} + v\tau_{yy} + kT_y \end{Bmatrix}, \tag{1.6}$$

where τ_{xx}, τ_{xy}, τ_{yx} and τ_{yy}, are the components of the stress tensor[1]

$$\begin{aligned} \tau_{xx} &= 2\mu\left(u_x - \tfrac{u_x + v_y}{3}\right), \\ \tau_{yy} &= 2\mu\left(v_y - \tfrac{u_x + v_y}{3}\right), \\ \tau_{xy} &= \tau_{yx} = \mu(v_x + u_y), \end{aligned} \tag{1.7}$$

where μ is the dynamic viscosity calculated using the Sutherland's law and k is the thermal conductivity.

[1]Note that we use Stokes hypothesis $\lambda = -2/3$.

1.3 Problem description

We aim to simulate the flow past a cylinder by solving the Compressible Navier Stokes equations. For our study we use the following free-stream parameters: A Mach number equal to $M_\infty = 0.2$, a Reynolds number $Re_{L=1} = 200$ and $Pr = 0.72$, with the pressure set to $p_\infty = 101325\ Pa$ and the density equal to $\rho = 1.225\ Kg/m^3$.

The flow domain is a rectangle of sizes [-10 20] x [-10 10]. The mesh consists of 639 quadrilaterals in which we applied the following boundary conditions (BCs): $Non - slip\ isothermal\ wall$ on the cylinder surface, $far - field$ at the bottom and top boundaries, $inflow$ at the left boundary and $outflow$ at the right boundary.

For the Navier-Stokes equations a $non - slip$ condition must me applied to the velocity field at a solid wall, which corresponds to the cylinder for this problem. The cylinder wall is defined as an isothermal wall with imposed temperature $T_{wall} = 300.15\ K$.

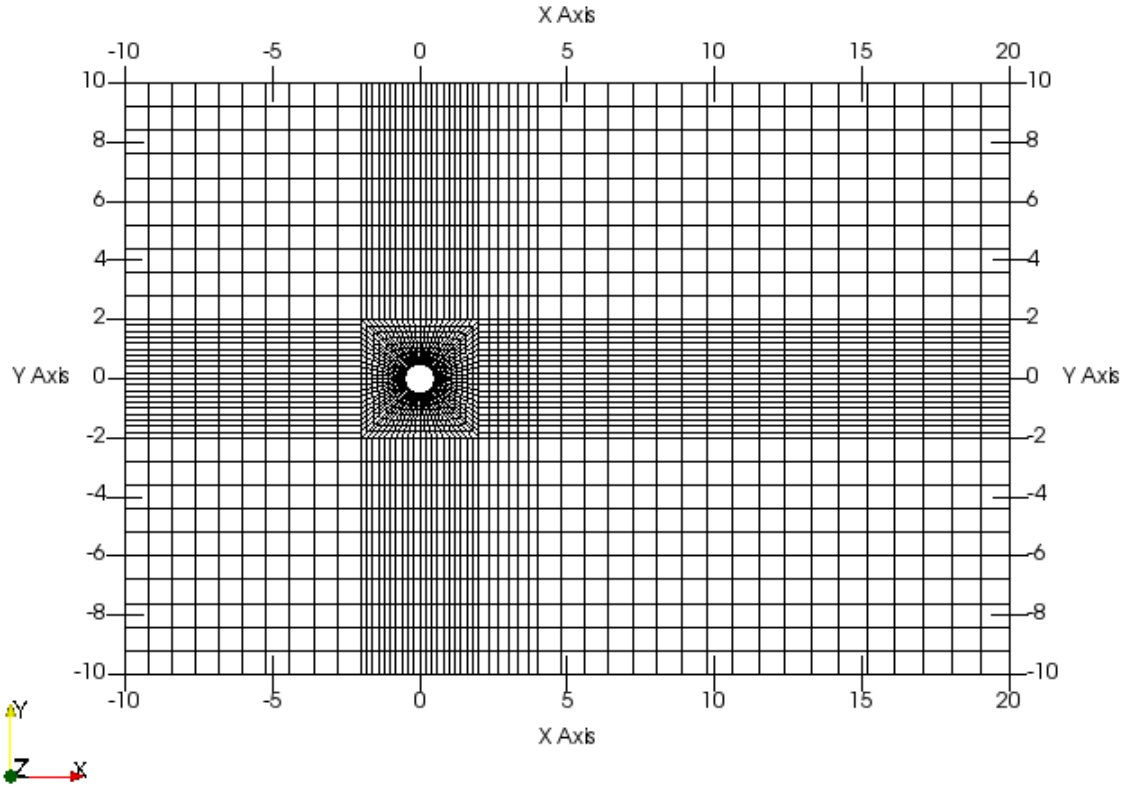

Figure 1.1 639 elements mesh.

$Inflow$, $Outflow$ and $Farfield$ BCs:

In the Compressible Flow Solver the boundary conditions are weakly implemented- (i.e the BCs are applied to the fluxes). In the Euler equations, for farfield BCs, the flux is

computed via a Riemann solver. The use of a Riemann solver for applying BCs implies the usage of a ghost point where it is necessary to apply a consistent ghost state, which is not always trivial. In evaluating the boundary, the Riemann solver takes automatically into account the eigenvalues (characteristic lines) of the Euler equations and therefore the problem is always well posed. This approach is equivalent to a characteristic approach where the corresponding Riemann invariants are computed and applied as BCs, taking into account if the boundary is an inflow or outflow. The method is also known as no-reflective BCs as it damps the spurious reflections from the boundaries.

The characteristic approach presented for the Euler equations for farfield boundaries, works also for the advective flux of the Navier-Stokes equations in regions where viscosity effects can be neglected. However, in our outflow case shedding is present, so viscosity effects become important. In this case, the characteristic treatment of the BCs generates spurious oscillations polluting the overall solution and leading to numerical instabilities. In order to avoid this, $Nektar++$ implements a method based on the so-called sponge terms, modifying the RHS of the compressible NS equations as follows:

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f_1}}{\partial x_1} + \frac{\partial \mathbf{f_2}}{\partial x_2} = \sigma(\bar{\mathbf{x}})(\mathbf{u_{ref}} - \mathbf{u}), \tag{1.8}$$

Where $\sigma(\bar{\mathbf{x}})$ is a damping coefficient defined in a region $\bar{\mathbf{x}}$ in proximity to the boundaries and u_{ref} is a known reference solution. The length and the shape of the damping coefficient depend on the problem being solved.

For further understanding of the boundary conditions implementation, please visit A Guide to the Implementation of Boundary Conditions in Compact High-Order Methods for Compressible Aerodynamics.

The initial condition is chosen to be that of a free flow field without the cylinder. If the solution greatly differs from the initial condition waves develop giving stability problems.

Tip

Set the initial conditions close to the expected solution to accelerate convergence and increment stability. Examples of setting more realistic initial conditions:

- In the case of a low Mach number, an incompressible flow solution can be used as initial condition.
- Also, setting an inviscid solution as initial conditions may help. Note that this can be done by selecting Euler equations instead of Navier-Stokes in the `SOLVERINFO` tag.

We successively setup the parameters of the problem (section 2.3). We finally run the solver (section 3) and post-process the data in order to visualise the results (section 4).

Chapter 2

Pre-processing

To set up the problem we have three steps. The first is setting up a mesh as discussed in section 2.1. The second one is setting the expansion bases as explained in section 2.2. We also need to configure the problem initial conditions, boundary conditions and parameters which are discussed in 2.3.

2.1 Mesh generation

The first pre-processing step consists in generating the mesh in a *Nektar++* compatible format. One option to do this is to use the open-source mesh-generator Gmesh to first create the geometry. The mesh format provided by Gmesh is not consistent with the *Nektar++* solvers and, therefore, it needs to be converted. An example of how to do this can be found in the Advection Solver Tutorial.

For two-dimensional simulations, the mesh definition contains 6 tags encapsulated within the `GEOMETRY` tag. The first tag, `VERTEX`, contains the spatial coordinates of the vertices of the various elements of the mesh. The second tag, `EDGE` contains the lines connecting the vertices. The third tag, `ELEMENT`, defines the elements (note that in this case we have only quadrilateral - e.g. `<Q ID="85">` - elements). The fourth tag, `CURVED`, is used to describe the control points for the curve. Note this tag is only necessary if curved edges or faces are present in the mesh and may otherwise be obmitted. The fifth tag, `COMPOSITE`, is constituted by the physical regions of the mesh called **composite**, where the composites formed by elements represent the solution sub-domains - i.e. the mesh sub-domains where we want to solve our set of equations (note that we will use these composites to define expansion bases on each sub-domain in section 2.3) - while the composites formed by edges are the boundaries of the domain where we need to apply suitable boundary conditions (note that we will use these composites to specify the boundary conditions in section 2.3). Finally, the sixth tag, `DOMAIN`, formally specifies the overall solution domain as the union of the composites forming the solution subdomains (note that the specification of different subdomain - i.e. composites - in this case is not necessary since they are constituted by same element shapes). For additional details on

the `GEOMETRY` tag refer to the User-Guide.

```
<?xml version="1.0" encoding="utf-8" ?>
<NEKTAR>
    <GEOMETRY DIM="2" SPACE="2">
        <VERTEX>
            <V ID="0">-1.00000000e+01 1.00000000e+01 0.00000000e+00 </V>
            ...
            <V ID="706">-4.93844170e-01 -7.82172325e-02 0.00000000e+00 </V>
        </VERTEX>
        <EDGE>
            <E ID="0"> 0 1 </E>
            ...
            <E ID="1346"> 706 668 </E>
        </EDGE>
        <ELEMENT>
            <Q ID="0"> 0 1 2 3 </Q>
            ...
            <Q ID="639"> 1345 1346 1269 615 </Q>
        </ELEMENT>
        <CURVED>
            <E ID="0" EDGEID="1344" NUMPOINTS="4" TYPE="PolyEvenlySpaced"> ...
            ...
            <E ID="1346" EDGEID="235" NUMPOINTS="4" TYPE="PolyEvenlySpaced"> ...
        </CURVED>
        <COMPOSITE>
            <C ID="100"> E[1268,1271,...,1344,1346] </C>
            <C ID="101"> E[3,6,...,1256,1266] </C>
            ...
            <C ID="0"> Q[0-639] </C>
        </COMPOSITE>
        <DOMAIN> C[0] </DOMAIN>
    </GEOMETRY>
</NEKTAR>
```

Note

In this case the mesh has been defined under the `GEOMETRY` tag with the `EXPANSIONS` definition and the `CONDITIONS` section in the same .xml file. However, the mesh can be a separate input .xml format containing only the geometry definition. Also, note that this mesh is in uncompressed format. In order to reduce the size of a large mesh compressed format should be used.

2.2 Expansion bases

We need to specify the expansion bases we want to use in each of the composites or sub-domains (`COMPOSITE=".."`) introduced in section 2.1:

```
<EXPANSIONS>
    <E COMPOSITE="C[0]" NUMMODES="3" FIELDS="rho,rhou,rhov,E"
    TYPE="MODIFIED" />
</EXPANSIONS>
```

For this case there is only one composite, `COMPOSITE="C[0]"`, where `NUMMODES` is the number of coefficients we want to use for the basis functions (that is commonly equal to P+1 where P is the polynomial order of the basis functions), `TYPE` allows selecting the basis functions, `FIELDS` is the solution variable of our problem and `COMPOSITE` are the mesh regions. For additional details on the `EXPANSIONS` tag refer to the User-Guide.

Tip

One source of instability is aliasing effects which arise from the nonlinearity of the underlying problem. Dealiasing techniques based on the concept of consistent integration can be applied in order to improve the robustness of the solver. For further information about dealisaing techniques, please check Dealiasing techniques for high-order spectral element methods on regular and irregular grids.

An example of dealiasing technique on for quadrilateral elements:

```
<EXPANSIONS>
        <E COMPOSITE="C[0]" BASISTYPE="GLL_Lagrange,GLL_Lagrange"
           NUMMODES="5,5" POINTSTYPE="GaussLobattoLegendre,GaussLobattoLegendre"
           NUMPOINTS="10,10" FIELDS="rho,rhou,rhov,E" />
</EXPANSIONS>
```

2.3 Configuring problem definitions

We will now proceed to set up the various problem parameters, the solver settings, initial and boundary conditions.

Parameters

The case will be run at Mach number equal to $M_\infty = 0.2$, Reynolds number $Re_{L=1} = 200$ and $Pr = 0.72$, with the pressure set to $p_\infty = 101325\ Pa$ and the density equal to $\rho = 1.225\ Kg/m^3$. The cylinder is defined as an isothermal wall with imposed temperature $T_{wall} = 300.15\ K$.

Within `PARAMETERS` tag, we can can also define the final physical time of the simulation, `FinTime`, the number of steps `NumSteps`, the step-interval when an output file is written `IO_CheckSteps` and the step-interval when information about the simulation is printed to the screen `IO_InfoSteps`.

Task 2.1

In the .xml file under the tag `PARAMETERS`, define all the flow parameters as described above. These are declared as $Mach$, Re, Pr, $pinf$, $rhoinf$ and $Twall$. Define the number of steps $NumSteps$ as the ratio of the $FinalTime$ to the time-step $TimeStep$.

Warning

Do not define both *Prandtl number* and the *thermal conductivity* parameters. They are correlated and defining both will prevent the simulation to start.

```
<PARAMETERS>
    <P> TimeStep = 0.00001 </P>
    <P> FinTime = 0.01 </P>
    <P> NumSteps = FinTime/TimeStep </P>
    <P> IO_CheckSteps = 100 </P>
    <P> IO_InfoSteps = 100 </P>
    <P> GasConstant = 287.058 </P>
    <P> Gamma = 1.4 </P>
    <P> pInf = 101325 </P>
    <P> rhoInf = 1.225 </P>
    <P> Mach = 0.2 </P>
    <P> cInf = sqrt(Gamma * pInf / rhoInf) </P>
    <P> uInf = Mach*cInf </P>
    <P> vInf = 0.0 </P>
    <P> Twall = 300.15 </P>
    <P> Re = 200 </P>
    <P> L = 1 </P>
    <P> mu = rhoInf * L * uInf / Re </P>
    <P> Pr = 0.72 </P>
  </PARAMETERS>
```

Solver Settings

We now declare how the flow will be solved. We want to include the effects of fluid viscosity and heat conduction and consequently the equation type we are going to use is the Navier-Stokes equations.

Note

In *Nektar++* the spatial discretization of the compressible Navier-Stokes equations is projected in the polynomial space via discontinuous projection. Specifically we make use of either of the discontinuous Galerkin (DG) method or the Flux-Reconstruction (FR) approach. Consequently, set the `Projection` to `DisContinuous`, as Continuous Projection is not supported in the Compressible Flow Solver.

We must specify the advection type which will be the classical DG in weak form. Note *Nektar++* also presents the FR_{DG} scheme, which recovers the DG scheme with exact mass matrix, the FR_{HU} scheme, which recovers the DG scheme with lumped mass matrix and the FR_{SD} scheme, which recovers a spectral difference scheme. We must also define the diffusion operator we want to use, which will be local Discontinuous Galerkin and the time integration method which will be the Classical Runge Kutta of order 4.

Tip

When selecting the Advection Type scheme, bear in mind that:

- The error associated with the FR_{DG} and $DG_{SEM} - EMM$ scheme is the lowest. It corresponds to the most accurate scheme but it also presents the most severe restrictions in terms of time-step.
- The FR_{HU} and FR_{SD} are slightly less accurate but have more favourable time-step restrictions.
- For futher understanding, please visit Connections between the discontinuous Galerkin method and high-order flux reconstruction schemes and On the Connections Between Discontinuous Galerkin and Flux Reconstruction Schemes: Extension to Curvilinear Meshes..

Additionally, we need to define the Upwind Type (i.e. Riemann solver) we want to use for the advection operator. For this problem we will use HLLC (Harten, Lax, van Leer+Contact) Riemann solver. Also, we will use the constant viscosity type.

Note

A Riemann problem is solved at each interface of the computational domain for the advection term. $Nektar++$ provides ten different Riemann solvers, one exact and nine approximated. The exact one solves the problem using a Newton iterative method. The high accuracy of this method may imply a high computational cost. The approximated Riemann solvers do not take into account the full Riemann problem, these simplifications of the exact solver provide lower computational cost but lower accuracy.

Task 2.2

In the .xml file under the tag `SOLVERINFO`, define all the solver parameters as described above. These are declared as EQType, Projection, AdvectionType, DiffusionType, TimeIntegrationMethod, UpwindType, ViscosityType.

```
<SOLVERINFO>
        <I PROPERTY="EQType" VALUE="NavierStokesCFE" />
        <I PROPERTY="Projection" VALUE="DisContinuous" />
        <I PROPERTY="AdvectionType" VALUE="WeakDG" />
        <I PROPERTY="DiffusionType" VALUE="LDGNS" />
        <I PROPERTY="TimeIntegrationMethod" VALUE="ClassicalRungeKutta4"/>
        <I PROPERTY="UpwindType" VALUE="HLLC" />
        <I PROPERTY="ProblemType" VALUE="General" />
        <I PROPERTY="ViscosityType" VALUE="Constant" />
</SOLVERINFO>
```

Variables

In the `VARIABLES` tag we set the solution variable. For the 2D case we have:

```
<VARIABLES>
      <V ID="0"> rho </V>
      <V ID="1"> rhou </V>
      <V ID="2"> rhov </V>
      <V ID="3"> E </V>
 </VARIABLES>
```

Note again the weak enforcement of the boundary conditions. The BCs are applied to the fluxes rather than to the non conservative variables of the problem. For further understanding, please check A guide to the Implementation of the Boundary Conditions.

Boundary Conditions

The `BOUNDARYREGIONS` tag specifies the regions where to apply the boundary conditions.

```
<BOUNDARYREGIONS>
      <B ID="0"> C[100] </B>
      <B ID="1"> C[101] </B>
      <B ID="2"> C[102] </B>
      <B ID="3"> C[103] </B>
</BOUNDARYREGIONS>
```

The next tag is `BOUNDARYCONDITIONS` by which the boundary conditions are actually specified for each boundary ID specified in the `BOUNDARYREGIONS` tag. The boundary conditions have been set as explained in section 1.3

```
<!-- Wall -->
   <REGION REF="0">
        <D VAR="rho" USERDEFINEDTYPE="WallViscous" VALUE="0" />
        <D VAR="rhou" USERDEFINEDTYPE="WallViscous" VALUE="0" />
        <D VAR="rhov" USERDEFINEDTYPE="WallViscous" VALUE="0" />
        <D VAR="E" USERDEFINEDTYPE="WallViscous" VALUE="0" />
   </REGION>
<!-- Farfield -->
      <REGION REF="1">
        <D VAR="rho" VALUE="rhoInf" />
        <D VAR="rhou" VALUE="rhoInf*uInf" />
        <D VAR="rhov" VALUE="rhoInf*vInf" />
        <D VAR="E" VALUE="pInf/(Gamma-1)+0.5*rhoInf*(uInf*uInf+vInf*vInf)" />
      </REGION>
<!-- Inflow -->
      <REGION REF="2">
        <D VAR="rho" VALUE="rhoInf" />
        <D VAR="rhou" VALUE="rhoInf*uInf" />
        <D VAR="rhov" VALUE="rhoInf*vInf" />
        <D VAR="E" VALUE="pInf/(Gamma-1)+0.5*rhoInf*(uInf*uInf+vInf*vInf)" />
      </REGION>


```

```
25 <!-- Outflow -->
26       <REGION REF="3">
27         <D VAR="rho" VALUE="rhoInf" />
28         <D VAR="rhou" VALUE="rhoInf*uInf" />
29         <D VAR="rhov" VALUE="rhoInf*vInf" />
30         <D VAR="E" VALUE="pInf/(Gamma-1)+0.5*rhoInf*(uInf*uInf+vInf*vInf)" />
31       </REGION>
```

Note

As explained in section 2.3 Continuous Projection is not supported in the Compressible Flow Solver. Therefore, boundary conditions are specified through Dirichlet BCs and Neumann BCs are not supported.

The initial conditions have been set as explained in section 1.3.

```
1 <FUNCTION NAME="InitialConditions">
2         <E VAR="rho" VALUE="rhoInf"/>
3         <E VAR="rhou" VALUE="rhoInf*uInf" />
4         <E VAR="rhov" VALUE="rhoInf*vInf" />
5         <E VAR="E" VALUE="pInf/(Gamma-1)+0.5*rhoInf*(uInf*uInf+vInf*vInf)"/>
6 </FUNCTION>
```

2.4 Artificial Viscosity

In order to stabilise the flow in the presence of flow discontinuities we utilise a shock capturing technique which makes use of artificial viscosity to damp oscillations in the solution, in conjunction with a discontinuity sensor to decide where the addition of artificial viscosity is needed.

Tip

In order to turn the NonSmooth artificial viscosity on:

- Include `ShockCaptureType` option in `SOLVERINFO` tag and set it to `NonSmooth`.
- Set the parameters `Skappa`, `Kappa` and `mu0` in the `PARAMETERS` tag. `mu0` is the maximum value for the viscosity, `Kappa` is half of the width of the transition interval and `SKappa` is value of the centre of the interval.

  The viscosity varies from 0 to the maximum values as the sensor goes from $Skappa$-$Kappa$ to $SKappa$+$Kappa$.
- The default values are: `Skappa`=-1.3; `kappa`=0.2; `mu0`=1.0.
- For futher details, please read chapter 3 of Mesh adaptation strategies for compressible flows using a high-order spectral/hp element discretisation

Chapter 3

Running the solver

The `CompressibleFlowSolver` can be run to solve the Cylinder Subsonic problem.

Task 3.1

Run the solver by typing the following command on the command line: `$NEK/CompressibleFlowSolver CylinderSubsonic_NS.xml`

Tip

To reduce the solution time on computers with multiple processors, MPI can be used to run the simulation in parallel. Note that, for binaries compiled from source, the Cmake option `NEKTAR_USE_MPI` must have been set `ON`. To run in parallel, prefix the command in the previous task with `mpirun -np X`, replacing `X` by the number of parallel processes to use. For example, to use 32 processes: `mpirun -np 32 $NEK/CompressibleFlowSolver CylinderSubsonic_NS.xml`

The simulation has now produced 10 `.chk` binary files and a final `.fld` binary file. These binary files contain the result of the simulation every 100 time-steps. This output interval has been chosen through the parameter `IO_CheckSteps` in `PARAMETERS` tag. Also, it is possible to note that every 100 time-steps the solver outputs the physical time of the simulation and the CPU time required for doing 100 time-steps. The interval of 100 time-steps is decided through the parameter `IO_InfoSteps`.

Tip

Stability plays a crucial role in the Compressible Flow solver. To ensure the solution is not polluted leading to numerical instabilities, for long simulations the `.chk` files can be checked before the simulation ends.

Chapter 4

Simulation Results

Now that the simulation has been completed, we need to post-process the file in order to visualise the results. In order to do so, we can use the built-in post-processing routines within $Nektar++$. In particular we can use the following command:

Task 4.1

Convert the .xml and .chk files into a .vtu format by calling `$NEK/FieldConvert CylinderSubsonic_NS.xml CylinderSubsonic_NS.fld CylinderSubsonic_NS.vtu`.

Which generates a `.vtu` file that is a readable format for the open-source package Paraview. We can now open the `.vtu` file just generated and visualise it with Paraview. If we want to monitor the evolution of the simulation we can make an animation in Paraview by converting successive .chk files into .vtu

Task 4.2

Set the `FinTime` to 0.6 and run the simulation. In order to do that, define the number of steps $NumSteps$ as the ratio of the $FinalTime$ to the time-step $TimeStep$ and set the $FinalTime$. Remember to use MPI in order to reduce the simulation time.

To create the animation we need to convert the .xml files into .vtu format. To avoid typing the same command several times, create a routine to create the different .vtu files. Once all the .vtu files are created (they are found in the completed folder), open them in paraview as a group (i.e File/Open and select all of them without expanding the tab).

If the final time is set to 0.6 and the .chk files are obtained every 400 steps. The animation created with the last 20 files should look like the CylinderSubsonic_NS.ogv video included in the completed folder.

Convert the .xml and .fld files into a .vtu format as shown in Task 4.1.

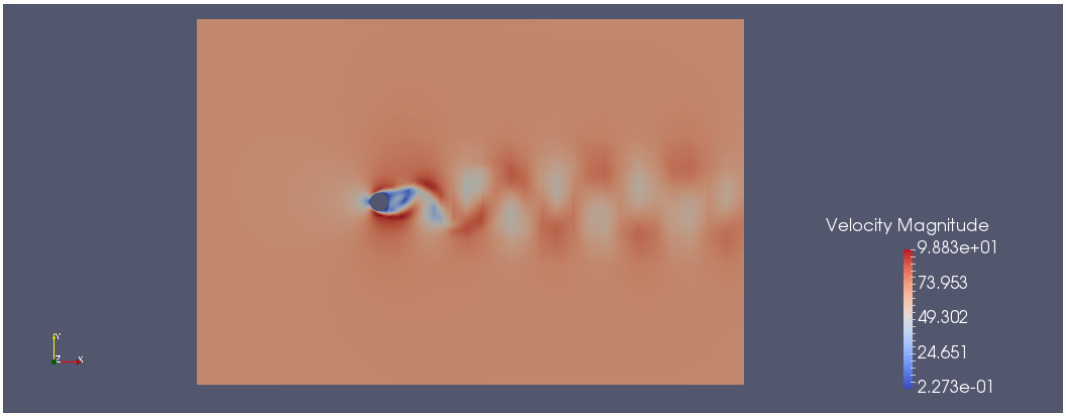

Figure 4.1 Instantaneous Velocity Flow Field

Calculate Vorticity

To perform the vorticity calculation and obtain an output data containing the vorticity solution, the user can run:

Task 4.3

Create a .fld file with the vorticity with the command:
`$NEK/FieldConvert -m vorticity CylinderSubsonic_NS.xml CylinderSubsonic_NS.fld CylinderSubsonic_NS_vort.fld`

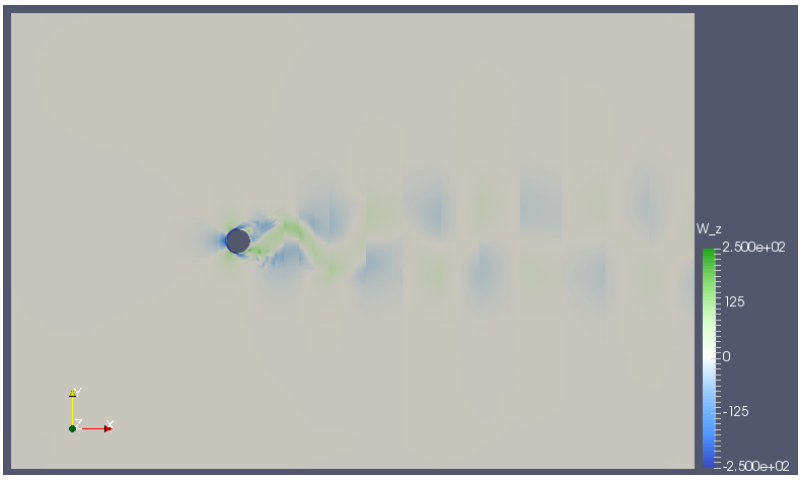

Figure 4.2 Instantaneous Vorticity Flow Field

Extract Wall Shear Stress

To obtain the wall shear stress vector and magnitude, the user can run:

```
FieldConvert -m wss:bnd=0:addnormals=0 CylinderSubsonic_NS.xml
CylinderSubsonic_NS.fld CylinderSubsonic_NS_wss.fld
```

The option `bnd` specifies which boundary region to extract. In this case the boundary region ID of the cylinder is 0. If the `addnormals` is turned on, $Nektar++$ additionally outputs the normal vector of the extracted boundary region.

In order to process the ouput file(s) you will need an .xml file of the same region. In order to do that we can use the NekMesh module extract:

```
NekMesh -m extract:surf=100 CylinderSubsonic_NS.xml bl.xml
```

Note, for NekMesh the surface ID we want to extract corresponds to the composite number of the cylinder surface -i.e 100.

To process the surface file one can use:

```
FieldConvert bl.xml CylinderSubsonic_NS_wss.fld
CylinderSubsonic_NS_wss.vtu
```

This command will generate a .dat file with the flow field information in the cylinder wall. It will produce the information of the density rho, the fluxes $rhou$, $rhov$ and E, the pressure p, the sound velocity a, the Mach number $Mach$, the sensor values $Sensor$, the shear values $Shear_x$, $Shear_y$ and $Shear_{mag}$ and the norms $norm_x$ and $norm_y$ for the different x and y coordinated along the cylinder. These files can be obtained from the completed folder.

This completes the tutorial.