OverBlown : A Fluid Flow Solver For Overlapping Grids, Reference Guide, Version 1.0

William D. Henshaw
Centre for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA, 94551.
henshaw@llnl.gov
http://www.llnl.gov/casc/people/henshaw
http://www.llnl.gov/casc/Overture

November 6, 2003

UCRL-MA-134289

Abstract:

OverBlown is a program that can be used to solve fluid flow problems on overlapping grids. It is built upon the **Overture** object-oriented framework. This is the **reference guide** for **OverBlown** . This document provides detailed information about the equations, discretizations and algorithms used. Refer to the **OverBlown** User Guide [10] for an introduction to **OverBlown** and its capabilities.
OverBlown has a number of different algorithms that can be used to solve problems for a range of Mach numbers. The Mach number, M, is the ratio of the flow speed to the speed of sound. In particular there are algorithms suited for

- incompressible flow, $M = 0$, (method INS)
- low Mach number flows, $M < .5$, (method ASF)
- moderate Mach numbers $.25 < M < 1.0$, (method CNS) and high Mach number flows $25 < M$, (method CNSCAD).
- reactive Euler equations in 2D (method CNSGOD).

OverBlown can be used to solve problems on moving grids. **OverBlown** can also be used to solve simple chemically reacting flows.

Contents

1 Introduction

OverBlown is a fluid flow solver for overlapping grids built upon the **Overture** framework [1],[5],[2]. **OverBlown** can be used to solve the incompressible Navier-Stokes equations (INS), and the compressible Navier-Stokes equations using either an all-speed flow algorithm (ASF), a moderate Mach number algorithm (CNS) or a high Mach number algorithm (CNSCAD). The ASF algorithm would be appropriate from low to moderate Mach number, say $M < .5$, the CNS algorithm best for $.25 < M < 1.$ while the CNSCAD algorithm is best for $M > .25$ (approximately).

More information about **Overture** can be found on the **Overture** home page, `http://www.llnl.gov/casc/Overture`. For installation procedures see the **OverBlown** user guide.
Other documents of interest that are available through the **Overture** home page are

- The **OverBlown** User Guide [10] shows how to run **OverBlown** and specify parameters.
- The overlapping grid generator, `Ogen`, [7]. Use this program to make grids for **OverBlown** .
- Mapping class documentation : `mapping.tex`, [6]. Many of the mappings that are used to create an overlapping grid are documented here.
- Interactive plotting : `PlotStuff.tex`, [9].
- `Oges` overlapping grid equation solver, used by **OverBlown** to solve implicit time stepping equations and the Poisson equation for the pressure, [8].

2 Structure of the OverBlown Code

Here is a brief overview of the Classes that make up **OverBlown** .

class OB_Parameters : Contains the parameters associated with **OverBlown** , such as

- **enum PDE pde** : holds the pde we are solving such as `incompressibleNavierStokes`.
- **enum TimeSteppingMethod timeSteppingMethod** : The time stepping method we are using such as `adamsPredictorCorrector2`.
- **real machNumber, reynoldsNumber,...** : parameters associated with the PDE's we know how to solve.
- **int numberOfComponents** : The number of 'components' in the equations; this would be 3 (u, v, p) for the 2D incompressible Navier-Stokes.
- **int rc,uc,vc,pc,...** : integers that indicate the positions of the density (rc), horizontal velocity (uc),... in the grid functions. For example for the compressible Navier Stokes the density is stored as `u(I1,I2,I3,rc)`

class OB_MappedGridSolver : contains methods associated with solving a PDE at the MappedGrid level.

- **method getUt** : generic routine for computing du/dt on a MappedGrid.
- **method getUtINS, getUtCNS,...** : compute du/dt for different equations, for example getUtINS is the routine for the incompressible Navier-Stokes.
- **method getTimeStep** : generic routine for computing the time step required for a component grid.
- **method getTimeSteppingEigenvalue** : generic routine for computing the time stepping "eigenvalue" from which the time stepping condition can be computed.
- **method getTimeSteppingEigenvalueINS,getTimeSteppingEigenvalueCNS,...** : compute time stepping eigenvalue for different PDEs.

class OB_CompositeGridFunction : container class holding a realCompositeGridFunction, a CompositeGrid, the time the grid function lives at and the grid velocity. This function also knows how to convert itself from primitive variables to conservative variables.

class OB_CompositeGridSolver : solvers on a CompositeGrid level. Includes method of lines time stepping routines and an Euler time step routine.

- **OB_CompositeGridFunction gf**[ ] : an array of grid functions used to hold different time levels required by the time stepping method.
- **method advance** : The main time stepping coordination routine to advance to a given final time, plot and save results, change the time step.
- **method printTimeStepInfo** : this is the function that prints information to the screen whenever the solution is plotted or saved in a file.
- **method advanceAdamsPredictorCorrector, advanceMidPoint,...** : advance the solution some number of steps using a particular method.
- **OB_MappedGridSolver** $**$**mappedGridSolver** : an array of pointers to solvers for each MappedGrid (different grids could use different solvers).

class OverBlown : controlling class for all of **OverBlown** .

- **setParameterValuesInteractively** : Main function for changing parameters.
- **setBoundaryConditionsInteractively** : specify boundary conditions.
- **getInitialConditions** : assign initial conditions.
- **solve** : solve the problem.

The main program (overBlown.C) looks something like

```
main()
{
  getFromADataBase(cg,nameOfOGFile);
  OverBlown solver(cg,&plotStuff,showFile,plotOption);

  solver.setParametersInteractively();

  solver.solve();

}
```

The `setParametersInteractively` function in OverBlown assigns parameters, boundary conditions and initial conditions,

```
OverBlown::setParametersInteractively()
{
  // choose a pde to solve

  // set PDE, runtime parameters, BC's

  // initialize solvers
  initialize();

  // get initial conditions
  compositeGridSolver[0]->initializeSolution()
}
```

The solve function in OverBlown just calls the advance function for the OB_CompositeGridSolver's, of which there is only one at the moment,

```
OverBlown::solve()
{
  compositeGridSolver[0]->advance(tFinal);
}
```

The CompositeGridSolver advance function:

```
OB_CompositeGridSolver::advance(tFinal)
{
  while( t<tFinal )
  {
    // output results
    plot(...);      // Here we wait for interactive changes
    saveShow(...);

    // compute the time step
    computeNumberOfStepsAndAdjustTheTimeStep(...);

   // advance numberOfSubSteps time steps :
   if( use predictor corrector )
      advanceAdamsPredictorCorrector(...);
   else if( use implicit )
      advanceImplicitMultiStep(...);
   else
     ...
  }
}
```

3 Time stepping methods

3.1 Adams Predictor-Corrector

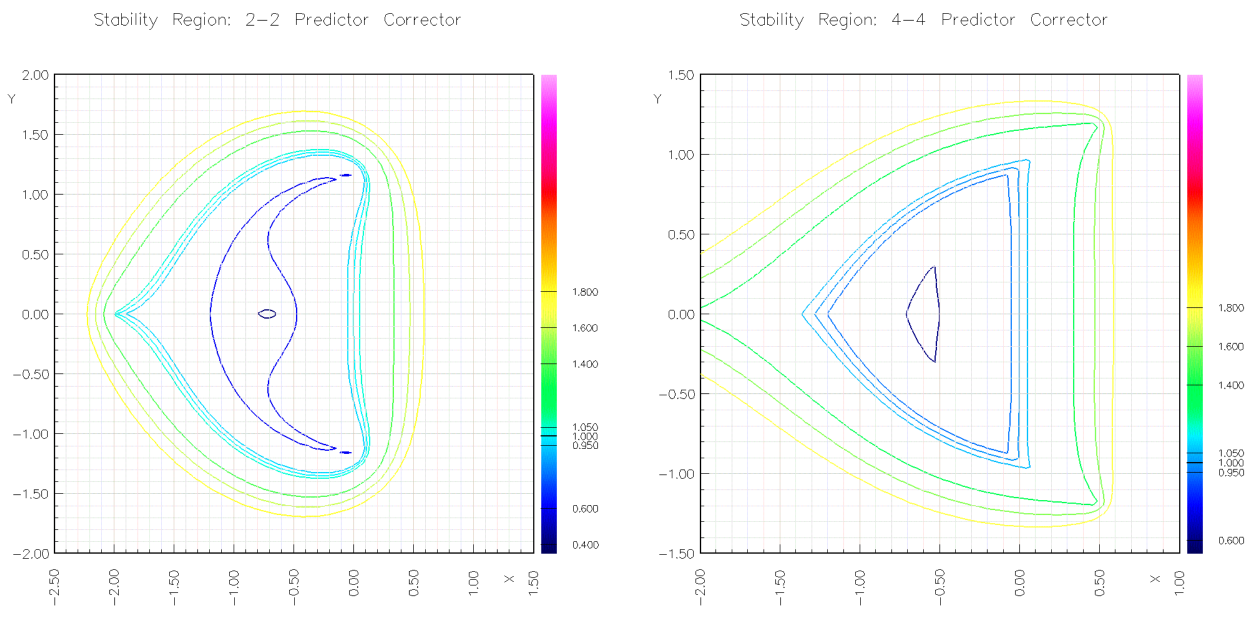

Figure 1: Stability regions for the predictor corrector methods. Left: second-order method, PECE mode. Right: fourth-order method, PECE mode.

If we write the INS equations as

$$\mathbf{u}_t = \mathbf{f}(\mathbf{u}, p)$$

where we may think of the pressure p as simply a function of $\mathbf{u}$ and we may use any time integrator in a method-of-lines fashion.

The **second-order accurate Adams predictor-corrector** time stepping method for the INS equations can be chosen with the “`adams PC`” option. It is defined by

$$\begin{aligned}\frac{\mathbf{u}^p - \mathbf{u}^n}{\Delta t} &= \frac{3}{2}\mathbf{f}^n - \frac{1}{2}\mathbf{f}^{n-1} \\ \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} &= \frac{1}{2}\mathbf{f}^p + \frac{1}{2}\mathbf{f}^n\end{aligned}$$

where we have shown one correction step (one may optionally correct more than one time). The stability region for this method is shown in figure (1).

To allow for a time-step that may change we actually use

$$\begin{aligned}\frac{\mathbf{u}^p - \mathbf{u}^n}{\Delta t} &= p_0\mathbf{f}^n + p_1\mathbf{f}^{n-1} \\ \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} &= \frac{1}{2}\mathbf{f}^p + \frac{1}{2}\mathbf{f}^n \\ p_0 &= 1 + \Delta t/(2\Delta t_1) \\ p_1 &= -\Delta t/(2\Delta t_1)\end{aligned}$$

where $\Delta t_1 = t_n - t_n - 1$.

The **fourth-order accurate Adams predictor-corrector** time stepping method for the INS equations can be chosen with the "`adams PC order 4`" option. It is defined by

$$\begin{aligned}
\frac{\mathbf{u}^p - \mathbf{u}^n}{\Delta t} &= \frac{1}{24}\Big[55\mathbf{f}^n - 59\mathbf{f}^{n-1} + 37\mathbf{f}_{n-2} - 9\mathbf{f}_{n-3}\Big] \\
\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} &= \frac{1}{24}\Big[9\mathbf{f}^p + 19\mathbf{f}^n - 5\mathbf{f}^{n-1} + \mathbf{f}_{n-2}\Big]
\end{aligned}$$

(see for example Lambert[**?**]). The stability region for this method is shown in figure (1).

To allow for a time-step that may change we actually use

$$\begin{aligned}
\frac{\mathbf{u}^p - \mathbf{u}^n}{\Delta t} &= p_0\mathbf{f}^n + p_1\mathbf{f}^{n-1} + p_2\mathbf{f}_{n-2} + p_3\mathbf{f}_{n-3} \\
\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} &= c_0\mathbf{f}^p + c_1\mathbf{f}^n + c_2\mathbf{f}^{n-1} + c_3\mathbf{f}_{n-2} \\
p_0 =& (6\Delta t_0\Delta t_2\Delta t_2 + 12\Delta t_2\Delta t_2\Delta t_1 + 8\Delta t_0\Delta t_0\Delta t_2 + 24\Delta t_2\Delta t_0\Delta t_1 + \\
& 12\Delta t_2\Delta t_1\Delta t_3 + 6\Delta t_3\Delta t_2\Delta t_0 + 24\Delta t_1\Delta t_1\Delta t_2 + 12\Delta t_0\Delta t_3\Delta t_1 + 18\Delta t_0\Delta t_1 \\
& \Delta t_1 + 4\Delta t_0\Delta t_0\Delta t_3 + 12\Delta t_1\Delta t_1\Delta t_3 + 3\Delta t_0\Delta t_0\Delta t_0 + 12\Delta t_0\Delta t_0\Delta t_1 + 12\Delta t_1 \\
& \Delta t_1\Delta t_1)\Delta t_0/(\Delta t_1 + \Delta t_2 + \Delta t_3)/\Delta t_1/(\Delta t_1 + \Delta t_2)/12 \\
p_1 =& -\Delta t_0\Delta t_0(6\Delta t_1\Delta t_1 + 6\Delta t_3\Delta t_1 + 12\Delta t_2\Delta t_1 + 8\Delta t_0\Delta t_1 + 3\Delta t_0 \\
& \Delta t_0 + 6\Delta t_2\Delta t_3 + 4\Delta t_0\Delta t_3 + 8\Delta t_2\Delta t_0 + 6\Delta t_2\Delta t_2)/\Delta t_1/(\Delta t_2 + \Delta t_3)/\Delta t_2/12 \\
p_2 =& \Delta t_0\Delta t_0(6\Delta t_1\Delta t_1 + 6\Delta t_2\Delta t_1 + 6\Delta t_3\Delta t_1 + 8\Delta t_0\Delta t_1 + 3\Delta t_0\Delta t_0 \\
& + 4\Delta t_2\Delta t_0 + 4\Delta t_0\Delta t_3)/\Delta t_3/\Delta t_2/(\Delta t_1 + \Delta t_2)/12 \\
p_3 =& -(6\Delta t_1\Delta t_1 + 6\Delta t_2\Delta t_1 + 8\Delta t_0\Delta t_1 + 4\Delta t_2\Delta t_0 + 3\Delta t_0\Delta t_0)\Delta t_0 \\
& \Delta t_0/(\Delta t_1 + \Delta t_2 + \Delta t_3)/(\Delta t_2 + \Delta t_3)/\Delta t_3/12 \\
c_0 =& (6\Delta t_1\Delta t_1 + 6\Delta t_2\Delta t_1 + 8\Delta t_0\Delta t_1 + 4\Delta t_2\Delta t_0 + 3\Delta t_0\Delta t_0) \\
& \Delta t_0/(\Delta t_0 + \Delta t_1 + \Delta t_2)/(\Delta t_0 + \Delta t_1)/12 \\
c_1 =& \Delta t_0(\Delta t_0\Delta t_0 + 4\Delta t_0\Delta t_1 + 2\Delta t_2\Delta t_0 + 6\Delta t_1\Delta t_1 + 6\Delta t_2\Delta t_1)/(\Delta t_1 + \Delta t_2)/\Delta t_1/12 \\
c_2 =& -\Delta t_0\Delta t_0\Delta t_0(\Delta t_0 + 2\Delta t_1 + 2\Delta t_2)/(\Delta t_0 + \Delta t_1)/\Delta t_2/\Delta t_1/12 \\
c_3 =& (\Delta t_0 + 2\Delta t_1)\Delta t_0\Delta t_0\Delta t_0/(\Delta t_0 + \Delta t_1 + \Delta t_2)/(\Delta t_1 + \Delta t_2)/\Delta t_2/12
\end{aligned}$$

where $\Delta t_m = t_{n+1-m} - t_{n-m}$, $m = 0, 1, 2, 3$.

3.2 Implicit multistep method

With the `implicit` time stepping method the INS equations are integrated with the viscous terms treated implicitly and the other terms treated with a 2nd-order Adams predictor corrector. If we split the equations into an explicit and implicit part,

$$\begin{aligned}
\mathbf{u}_t &= [-(\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla p] + \nu\Delta\mathbf{u} \\
\mathbf{u}_t &= \mathbf{f}_E + A\mathbf{u} \\
\mathbf{f}_E &= -(\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla p \\
A\mathbf{u} &= \nu\Delta\mathbf{u}
\end{aligned}$$

then the time step consists of a predictor,

$$\frac{\mathbf{u}^p - \mathbf{u}^n}{\Delta t} = \frac{3}{2}\mathbf{f}_E^n - \frac{1}{2}\mathbf{f}_E^{n-1} + \alpha A\mathbf{u}^p + (1-\alpha)A\mathbf{u}^n$$

and a corrector

$$\frac{\mathbf{u}^c - \mathbf{u}^n}{\Delta t} = \frac{1}{2}\mathbf{f}_E^p + \frac{1}{2}\mathbf{f}_E^n + \alpha A\mathbf{u}^c + (1-\alpha)A\mathbf{u}^n$$

The `implicit factor` α can be set as a parameter. A value of $\alpha = \frac{1}{2}$ will give a second-order Crank-Nicolson method. A value of $\alpha = 1$ will give a first-order backward-Euler method.

3.3 Variable time stepping

The `variableTimeStepAdamsPredictorCorrector` time stepping option allows each grid to have it's own time step.

4 Equations

4.1 Method INS: Incompressible Navier-Stokes Equations

See the OverBlownINS document [**?**] for a description of the OverBlown's incompressible Navier-Stokes solver.

4.2 Compressible Navier-Stokes Equations

OverBlown can solve the compressible Navier-Stokes equations. Currently there are four different methods available to solve these equations,

1. Method CNSCAD: Solve the equations in conservation form with a conservative discretization.
2. Method CNSGOD: Solve the Euler equations iin 2D with a Godnuov method.
3. Method CNS: Solve the equations in non-conservative form
4. Method ASF: An all-speed flow algorithm valid for low Mach number flows.

4.3 Method CNSCAD: Compressible Navier-Stokes equations using a conservative discretization

In this section we describe the method that solves the compressible Navier-Stokes and Euler equations with a conservative discretization.

Currently the code is second order accurate in two and three space dimensions. The discretization is conservative and vertex centred. The artificial viscosity we use is based on that developed by Jameson [15]. Interpolation is not done in a conservative fashion although in principle this could be done, as described in [3]. Future plans include incorporating some flux limiting methods as an alternative to artificial viscosity and some fourth-order accurate methods.

The compressible Navier-Stokes equations can be written in conservation form

$$\mathbf{u}_t + \frac{\partial \mathbf{F}}{\partial x_1} + \frac{\partial \mathbf{G}}{\partial x_2} + \frac{\partial \mathbf{H}}{\partial x_3} = 0.$$

The vector of conserved variables $\mathbf{u}$ is

$$\mathbf{u} = \begin{bmatrix} \rho \\ E \\ \rho v_1 \\ \rho v_2 \\ \rho v_3 \end{bmatrix},$$

where ρ, E and $\mathbf{v} = [v_1, v_2, v_3]^T$ denote the density, energy and velocity vector with components parallel to the x_1, x_2, and x_3 axes, respectively. The fluxes

$$\begin{bmatrix} \mathbf{F} \\ \mathbf{G} \\ \mathbf{H} \end{bmatrix} = \begin{bmatrix} \mathbf{F}^C \\ \mathbf{G}^C \\ \mathbf{H}^C \end{bmatrix} - \begin{bmatrix} \mathbf{F}^V \\ \mathbf{G}^V \\ \mathbf{H}^V \end{bmatrix}$$

are a combination of *convective* fluxes $[\mathbf{F}^C, \mathbf{G}^C, \mathbf{H}^C]^T$ and *viscous* fluxes $[\mathbf{F}^V, \mathbf{G}^V, \mathbf{H}^V]^T$. The convective (or Euler) fluxes are given by

$$\mathbf{F}^C = \begin{bmatrix} \rho v_1 \\ v_1(E+p) \\ \rho v_1^2 + p \\ \rho v_2 v_1 \\ \rho v_3 v_1 \end{bmatrix}, \ \mathbf{G}^C = \begin{bmatrix} \rho v_2 \\ v_2(E+p) \\ \rho v_1 v_2 \\ \rho v_2^2 + p \\ \rho v_3 v_2 \end{bmatrix}, \ \mathbf{H}^C = \begin{bmatrix} \rho v_3 \\ v_3(E+p) \\ \rho v_1 v_3 \\ \rho v_2 v_3 \\ \rho v_3^2 + p \end{bmatrix},$$

and the viscous fluxes are

$$\mathbf{F}^V = \begin{bmatrix} 0 \\ \sum_n v_n \tau_{1n} - q_1 \\ \tau_{11} \\ \tau_{12} \\ \tau_{13} \end{bmatrix}, \ \mathbf{G}^V = \begin{bmatrix} 0 \\ \sum_n v_n \tau_{2n} - q_2 \\ \tau_{21} \\ \tau_{22} \\ \tau_{23} \end{bmatrix}, \ \mathbf{H}^V = \begin{bmatrix} 0 \\ \sum_n v_n \tau_{3n} - q_3 \\ \tau_{31} \\ \tau_{32} \\ \tau_{33} \end{bmatrix}.$$

The pressure, p, and temperature, T, are given by the relations

$$\begin{aligned} p &= (\gamma - 1)[E - \frac{1}{2}\rho(v_1^2 + v_2^2 + v_3^2)], \\ T &= \frac{p}{\rho R_g}, \end{aligned}$$

where R_g is the gas constant. The viscous stress terms, τ_{mn}, and heat flux, q_n, are given by

$$\begin{aligned}\tau_{mn} &= \mu(\frac{\partial v_n}{\partial x_m} + \frac{\partial v_m}{\partial x_n}) - \frac{2}{3}\mu(\nabla\cdot\mathbf{v})\delta_{mn}, \\ q_n &= -k\frac{\partial T}{\partial x_n} = -\frac{k}{R_g}\frac{\partial}{\partial x_n}\left(\frac{p}{\rho}\right) := -\tilde{k}\frac{\partial}{\partial x_n}\left(\frac{p}{\rho}\right),\end{aligned}$$

where δ_{mn} is the Kronecker delta and $\tilde{k} = k/R_g$ is a scaled thermal conductivity.

4.3.1 Discretizing the equations

In this section the discretization of the equations is described. Let $\mathbf{x} = \mathbf{d}(\mathbf{r})$ denote a smooth mapping from the unit square, $\mathbf{r} = (r_1, r_2, r_3)$, into some portion of the physical domain. This mapping will define the region covered by a component grid. In the curvilinear coordinates, (r_1, r_2, r_3), the equations can be written in the conservation form

$$\mathbf{u}_t + \frac{1}{J}(\frac{\partial \mathbf{f}}{\partial r_1} + \frac{\partial \mathbf{g}}{\partial r_2} + \frac{\partial \mathbf{h}}{\partial r_3}) = 0,$$

where

$$\begin{aligned}\mathbf{f} &= a_{11}\mathbf{F} + a_{12}\mathbf{G} + a_{13}\mathbf{H}, \\ \mathbf{g} &= a_{21}\mathbf{F} + a_{22}\mathbf{G} + a_{23}\mathbf{H}, \\ \mathbf{h} &= a_{31}\mathbf{F} + a_{32}\mathbf{G} + a_{33}\mathbf{H}, \\ J &= \det\left[\frac{\partial \mathbf{x}}{\partial \mathbf{r}}\right].\end{aligned}$$

The Jacobian J is the determinant of the Jacobian matrix $\partial\mathbf{x}/\partial\mathbf{r}$. The transformation matrix $A = [a_{mn}]$ is given by

$$[a_{mn}] = J\left[\frac{\partial \mathbf{r}}{\partial \mathbf{x}}\right] = J\left[\frac{\partial \mathbf{x}}{\partial \mathbf{r}}\right]^{-1}.$$

For example

$$a_{11} = \det\begin{bmatrix} \frac{\partial x_2}{\partial r_2} & \frac{\partial x_2}{\partial r_3} \\ \frac{\partial x_3}{\partial r_2} & \frac{\partial x_3}{\partial r_3}\end{bmatrix}, \quad a_{12} = -\det\begin{bmatrix} \frac{\partial x_1}{\partial r_2} & \frac{\partial x_1}{\partial r_3} \\ \frac{\partial x_3}{\partial r_2} & \frac{\partial x_3}{\partial r_3}\end{bmatrix} \quad \ldots$$

To obtain the two-dimensional results set $\partial x_3/\partial r_n = \delta_{3n}$.

These equations are discretized with finite differences. Let $\mathbf{U}_i$ denote the discrete approximation to $\mathbf{u}(\mathbf{x}_i, t)$, and $\mathbf{V}_i = [V_{1,i}, V_{2,i}, V_{3,i}]^T$ the discrete approximation to the velocity $\mathbf{v}(\mathbf{x}_i, t)$. Here $i = (i_1, i_2, i_3)$ is a multi-index. For convenience $\mathbf{U}_{i_1+1}$ will be used to denote $\mathbf{U}_{i_1+1,i_2,i_3}$, $\mathbf{U}_{i_2+1}$ will denote $\mathbf{U}_{i_1,i_2+1,i_3}$, and so on. In addition to the convective and viscous fluxes we will also add an artificial viscosity with fluxes denoted by $[\mathbf{f}^A, \mathbf{g}^A, \mathbf{h}^A]^T$. The numerical approximation is given as

$$\begin{aligned}\frac{d}{dt}\mathbf{U}_i &+ \frac{1}{J_i}[D_{0r_1}\mathbf{f}_i^C + D_{0r_2}\mathbf{g}_i^C + D_{0r_3}\mathbf{h}_i^C] \\ &- \frac{1}{J_i}[D_{-r_1}\mathbf{f}^V_{i_1+\frac{1}{2}i_2} + D_{-r_2}\mathbf{g}^V_{i_2+\frac{1}{2}} + D_{-r_3}\mathbf{h}^V_{i_3+\frac{1}{2}}] \\ &- \frac{1}{J_i}[\Delta_{-r_1}\mathbf{f}^A_{i_1+\frac{1}{2}i_2} + \Delta_{-r_2}\mathbf{g}^A_{i_2+\frac{1}{2}} + \Delta_{-r_3}\mathbf{h}^A_{i_3+\frac{1}{2}}] = 0.\end{aligned}$$

The difference operators are defined as

$$D_{0r_1}\mathbf{U}_i = \frac{\mathbf{U}_{i_1+1}-\mathbf{U}_{i_1-1}}{2\Delta r_1}\ ,\ D_{-r_1}\mathbf{U}_i \ = \ \frac{\mathbf{U}_{i_1}-\mathbf{U}_{i_1-1}}{\Delta r_1}\ ,\ D_{+r_1}\mathbf{U}_i = \frac{\mathbf{U}_{i_1+1}-\mathbf{U}_{i_1}}{\Delta r_1}$$
$$\Delta_{+r_1}\mathbf{U}_i = \mathbf{U}_{i_1+1}-\mathbf{U}_{i_1} \quad , \quad \Delta_{-r_1}\mathbf{U}_i = \mathbf{U}_{i_1}-\mathbf{U}_{i_1-1}$$

The entries in the Jacobian matrix $\partial\mathbf{x}/\partial\mathbf{r}$ are assumed to be known at the grid points. The discrete convective fluxes are given by

$$\begin{aligned}
\mathbf{f}_i^C &= a_{11,i}\mathbf{F}(\mathbf{U}_i)+a_{12,i}\mathbf{G}(\mathbf{U}_i)+a_{13,i}\mathbf{H}(\mathbf{U}_i),\\
\mathbf{g}_i^C &= a_{21,i}\mathbf{F}(\mathbf{U}_i)+a_{22,i}\mathbf{G}(\mathbf{U}_i)+a_{23,i}\mathbf{H}(\mathbf{U}_i),\\
\mathbf{h}_i^C &= a_{31,i}\mathbf{F}(\mathbf{U}_i)+a_{32,i}\mathbf{G}(\mathbf{U}_i)+a_{33,i}\mathbf{H}(\mathbf{U}_i).
\end{aligned}$$

The definitions of $[\mathbf{f}^V_{i_1+\frac{1}{2}},\mathbf{g}^V_{i_2+\frac{1}{2}},\mathbf{h}^V_{i_3+\frac{1}{2}}]^T$ and $[\mathbf{f}^A_{i_1+\frac{1}{2}},\mathbf{g}^A_{i_2+\frac{1}{2}},\mathbf{h}^A_{i_3+\frac{1}{2}}]^T$ are described below.

4.3.2 True viscosity

The viscous fluxes, $[\mathbf{f}^V_{i_1+\frac{1}{2}},\mathbf{g}^V_{i_2+\frac{1}{2}},\mathbf{h}^V_{i_3+\frac{1}{2}}]$, are computed so that the resulting difference formula for $D_{-r_1}\mathbf{f}^V_{i_1+\frac{1}{2}} + D_{-r_2}\mathbf{g}^V_{i_2+\frac{1}{2}} + D_{-r_3}\mathbf{h}^V_{i_3+\frac{1}{2}}$ is a compact $3\times 3\times 3$ stencil. Consider for example the flux associated with the energy equation in the r_1 direction

$$f_2^V = a_{21}F_2^V + a_{22}G_2^V + a_{23}H_2^V,$$

where, for example,

$$\begin{aligned}
F_2^V &= \sum_{n=1}^{3} v_n\tau_{1n} - q_1,\\
\tau_{1n} &= \mu\Big(\frac{\partial v_n}{\partial x_1}+\frac{\partial v_1}{\partial x_n}\Big) - \frac{2}{3}\mu(\nabla\cdot\mathbf{v})\delta_{1n},\\
q_1 &= -\tilde{k}\frac{\partial}{\partial x_1}\left(\frac{p}{\rho}\right),
\end{aligned}$$

This term is discretized as

$$f^V_{2,i_1+\frac{1}{2}} = a_{21,i_1+\frac{1}{2}}F^V_{2,i_1+\frac{1}{2}} + a_{22,i_1+\frac{1}{2}}G^V_{2,i_1+\frac{1}{2}} + a_{23,i_1+\frac{1}{2}}H^V_{2,i_1+\frac{1}{2}},$$

where

$$a_{mn,i_1+\frac{1}{2}} = \frac{1}{2}(a_{mn,i_1+1}+a_{mn,i}),$$

and, for example,

$$F^V_{2,i_1+\frac{1}{2}} = V_{1,i_1+\frac{1}{2}}\tau_{11,i_1+\frac{1}{2}} + V_{2,i_1+\frac{1}{2}}\tau_{12,i_1+\frac{1}{2}} + V_{3,i_1+\frac{1}{2}}\tau_{13,i_1+\frac{1}{2}} - Q_{1,i_1+\frac{1}{2}},$$

where

$$\begin{aligned}
V_{n,i_1+\frac{1}{2}} &= \frac{1}{2}(V_{n,i_1+1}+V_{n,i}), \\
\tau_{11,i_1+\frac{1}{2}} &= \frac{4}{3}\mu\Big(D_{+x_1}V_{1,i}\Big) - \frac{2}{3}\mu\Big(\sum_{m=2}^{3} D_{+x_m}V_{mi}\Big), \\
\tau_{12,i_1+\frac{1}{2}} &= \mu\Big(D_{+x_1}V_{2,i} + D_{+x_2}V_{1,i}\Big), \\
\tau_{13,i_1+\frac{1}{2}} &= \mu\Big(D_{+x_1}V_{3,i} + D_{+x_3}V_{1,i}\Big), \\
Q_{i_1+\frac{1}{2}} &= -\tilde{k}D_{+x_1}\left(\frac{P_i}{R_i}\right).
\end{aligned}$$

Here, P_i is the discrete pressure and R_i is the discrete density, and the difference operator D_{+x_m} is defined by

$$D_{+x_m}W_i := \left(\frac{\partial r_1}{\partial x_m}\right)_{i_1+\frac{1}{2}} D_{+r_1}W_i + \sum_{n=2}^{3}\left(\frac{\partial r_n}{\partial x_m}\right)_{i_1+\frac{1}{2}} D_{+r_n}W_{i_1+\frac{1}{2}},$$

where W_i is any mesh function and

$$\left(\frac{\partial r_n}{\partial x_m}\right)_{i_1+\frac{1}{2}} = \frac{a_{nm,i_1+\frac{1}{2}}}{J_{i_1+\frac{1}{2}}} .$$

4.3.3 Artificial viscosity

If one is interested in solving the Euler equations ($\mu = 0$, $\tilde{k} = 0$) then it will be necessary to add artificial viscosity to stabilize the method and to give sharp shocks without overshoots. The artificial viscosity we use is similar to that developed by Jameson [15]. A good artificial viscosity has the property that a shock is free from numerical oscillations and is always just a few grid points wide. Ideally the parameters defining the artificial viscosity are independent of the the grid and independent of the particular solution. With our current artificial viscosity the parameters must still be changed when the Mach number changes significantly. In principle we believe we know to correct this problem, using suggestions from Professor Kreiss.

There are two components to the artificial viscosity, a second order term that is turned on near shocks and a fourth order term that is turned on away from shocks. For example, in the r_1 direction

$$\mathbf{f}^A_{i_1+\frac{1}{2}} = \mathbf{f}^2_{i_1+\frac{1}{2}} + \mathbf{f}^4_{i_1+\frac{1}{2}} .$$

The second-order artficial viscosity is

$$\mathbf{f}^2_{i_1+\frac{1}{2}} = \epsilon_2\lambda_{1,i_1+\frac{1}{2}}\min(1, s_{i_1+\frac{1}{2}}/w_2)\Delta_{+r_1}\mathbf{U}_i ,$$

and the fourth-order artificial viscosity is

$$\mathbf{f}^4_{i_1+\frac{1}{2}} = \epsilon_4\lambda_{1,i_1+\frac{1}{2}}\max(0, 1 - s_{i_1+\frac{1}{2}}/w_4)\Delta_{+r_1}(\Delta_{+r_1}\Delta_{-r_1}\mathbf{U}_i) .$$

The parameters ϵ_2, w_2, ϵ_4, w_4 are constants to be chosen. The parameter $s_{i_1+\frac{1}{2}}$ is a switch based on the variation of the pressure,

$$\begin{aligned}
s_{i_1+\frac{1}{2}} &= \max(S_{i_1-1}, S_{i_1}, S_{i_1+1}, S_{i_1+2}), \\
S_i &= \left|\frac{p_{i_1+1} - 2p_{i_1} + p_{i_1-1}}{p_{i_1+1} + 2p_{i_1} + p_{i_1-1}}\right|,
\end{aligned}$$

and $\lambda_{1,i_1+\frac{1}{2}}$ is proportional to the largest eigenvalue of the convective flux in the r_1 direction,

$$\lambda_{1,i_1+\frac{1}{2}} = \left\{ \Big|\sum_{n=1}^{3} a_{1n,i_1+\frac{1}{2}} V_{n,i_1+\frac{1}{2}}\Big| + c_{i_1+\frac{1}{2}} \Big(\sum_{n=1}^{3} a^2_{1n,i_1+\frac{1}{2}}\Big)^{\frac{1}{2}} \right\} \frac{1}{\Delta r_1}. \tag{1}$$

Here, any variables evaluated at $i_1+\frac{1}{2}$ are averaged and $c_{i_1+\frac{1}{2}}$ is the local speed of sound defined in terms of the discrete pressure P_i and discrete density R_i as

$$c^2_{i_1+\frac{1}{2}} = \frac{\gamma P_{i_1+\frac{1}{2}}}{R_{i_1+\frac{1}{2}}} \ .$$

The components of the artificial viscosity in the other directions are defined in an analogous fashion.

Choosing values for the coefficients in the artificial viscosity: The parameter w_2 is chosen so that $s_{i_1+\frac{1}{2}}/w_2 \geq 1$ for grid points i in the shock. This ensures that the second order term is turned on near the shock. Similarly, if the fourth order term is to be off near the shock, we want $w_4 \approx w_2$. The parameters ϵ_2 and ϵ_4 determine the size of each term. For shocks with Mach number about 2 we have found that the following values give a shock that extends over about 5 grid points without overshoots:

$$\epsilon_2 = \frac{1}{4} \quad , \quad w_2 = \frac{1}{120} \quad , \quad \epsilon_4 = \frac{1}{10} \quad , \quad w_4 = \frac{1}{100} \ .$$

These parameters were determined by trial and error by running the code on a one-dimensional shock problem.

BETAT,BETAK,RT0: Both μ and $\tilde{k}$ can be made to depend on the temperature. We have currently implemented a temperature dependence defined by

$$\mu(T) = \mu \left(\frac{R_g T}{R_g T_0}\right)^{\beta_T} \quad , \quad k(T) = k \left(\frac{R_g T}{R_g T_0}\right)^{\beta_k}$$

The parameters `betat` $= \beta_T$, `betak` $= \beta_k$ and `rt0` $= R_g T_0$ can be chosen by the user to define

$$\mu_i = \texttt{amu} \left(\frac{P_i/R_i}{\texttt{rt0}}\right)^{\texttt{betat}} \quad , \quad \tilde{k}_i = \texttt{akappa} \left(\frac{P_i/R_i}{\texttt{rt0}}\right)^{\texttt{betak}}$$

where P_i and R_i are the discrete pressure and density at grid point i. By default `betat=betak=0` so there is no dependence on the temperature.

AV2,AW2,AV4,AW4: These are the parameters in the artificial viscosity, `av2` $= \epsilon_2$, `aw2` $= w_2$, `av4` $= \epsilon_4$, and `aw4` $= w_4$.

4.4 Method CNSGOD: Reactive Euler equations with a Godunov Method

Together with Professor Don Schwendeman at RPI we have developed the capability in **OverBlown** for solving the reactive Euler equations on overlapping grids in two-dimensions. Very fine grids are required to resolve the structure of the detonation fronts that form and thus we use adaptive mesh refinement. The governing equations for a reaction involving n-species are

$$\mathbf{u}_t + \mathbf{f}(\mathbf{u})_x + \mathbf{g}(\mathbf{u})_y = \mathbf{h}(\mathbf{u})$$

with

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \\ \rho\lambda \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ u(\rho E + p) \\ \rho u \lambda \end{bmatrix},$$

$$\mathbf{g} = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ v(\rho E + p) \\ \rho v \lambda \end{bmatrix}, \quad \mathbf{h} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \rho R \end{bmatrix}$$

where $E = e + \frac{1}{2}(u^2 + v^2) - \sum_i^n \lambda_i Q_i$ and $e = e(\rho, p)$ is the equation of state. Here $\lambda_i, \;\; i = 1, 2, \ldots, n$ are the mass fractions and the Q_i represent the heat released. The one-step reaction model is given by

$$\begin{aligned} F &\xrightarrow{k_C} P, \quad \text{(Fuel to Product)}, \\ R(T, \lambda) &= (1 - \lambda) k_C, \\ k_C &= \sigma \exp(\frac{1}{\epsilon}(\frac{1}{T_C} - \frac{1}{T})), \end{aligned}$$

where λ is the mass fraction of product.

The equations are solved with a second-order accurate Godunov algorithm. The method itself is implemented in a fortran subroutine that is called from C++. The reaction equations are solved by sub-cycling, that is they use a smaller time step to remain stable and accurate. The error estimator is based on a combination of estimates of the spatial derivatives and an estimate of the reaction rate. The later term is important since reaction fronts can sometimes propogate rapidly through the grid and thus it may be necessary to refine in regions some distance from the current front. We typically use two levels of refinement factor of 4 with the AMR grids are regenerated every 4 or 8 steps depending on the size of the buffer zone.

4.5 Method CNS : Compressible Navier-Stokes equations using a non-conservative discretization

In this section we describe the method that solves the compressible Navier-Stokes equations written in a nonconservative fashion.

The equations have been discretized in both 2D and 3D to second-order accuracy in space.

When the Mach number is low and there are no shocks a nonconservative discretization of the equations is probably just as good as a conservative discretization. Moreover it is easier to implement higher-order accurate schemes in this form. Even for problems with shocks these equations give reasonable results - provided the shocks are smoothed out over 4 or 5 grid points any problems with conservation seem to go away.

The compressible Navier-Stokes equations for an ideal gas can be written in the the form

$$\rho_t + (\mathbf{v}\cdot\nabla)\rho + \rho(\nabla\cdot\mathbf{v}) = 0$$

$$\mathbf{v}_t + (\mathbf{v}\cdot\nabla)\mathbf{v} + \frac{1}{\rho}\nabla p - \frac{\mu}{\rho}[\Delta\mathbf{v} + \frac{1}{3}\nabla(\nabla\cdot\mathbf{v})] = 0$$

$$T_t + (\mathbf{v}\cdot\nabla)T + (\gamma-1)T\nabla\cdot\mathbf{v} - (\gamma-1)\frac{\tilde{k}}{\rho}\Delta T - (\gamma-1)\frac{\mu}{R_g\rho}\Phi = 0$$

where ρ is the density, $\mathbf{v} = [v_1, v_2, v_3]^T$ is the velocity, p the pressure and T the temperature. The equation of state for an ideal gas is

$$p = \rho R_g T$$

and

$$\Phi = \mu\sum_{ij}(v_i)_{x_j}((v_j)_{x_i} + (v_i)_{x_j}) + \lambda(\nabla\cdot\mathbf{v})^2$$

with $\lambda = -\frac{2}{3}\mu$. Also

$$\begin{aligned} \mu &= \text{viscosity coefficient} \\ R_g &= p/(\rho T),\ \text{gas constant} \\ \gamma &= C_p/C_v,\ \text{ratio of specific heats} \\ \tilde{k} &= k/R_g,\ \text{Coefficient of thermal conductivity over } R_g \end{aligned}$$

In non-dimensional units (with ρ,$\mathbf{v}$,T scaled to one) there are three nondimensional parameters, Re (Reynolds number), M (Mach number) and Pr (Prandtl number). They are related to μ, $\tilde{k}$, and R_g as:

$$\mu = \frac{1}{Re} \quad , \quad R_g = \frac{1}{\gamma M^2} \quad , \quad \tilde{k} = \frac{\gamma}{\gamma-1}\frac{1}{Pr}\frac{1}{Re}$$

4.5.1 Discretization of the Equations

The equations are discretized in space using standard finite-difference methods on overlapping grids. Associated with each component grid (numbered $k = 1, 2, \ldots, n_g$) there is a transformation, $\mathbf{d}_k$, that maps the unit cube, with coordinates denoted by $\mathbf{r} = (r_1, r_2, r_3)$, into physical space, $\mathbf{x} = (x_1, x_2, x_3)$,

$$\mathbf{x}(\mathbf{r}) = \mathbf{d}_k(\mathbf{r}) \ .$$

Each component grid, G_k, consists of a set of grid points,

$$G_k = \{\mathbf{x}_{i,k} \mid i = (i_1, i_2, i_3) \quad n_{m,a,k} - 2 \le i_m \le n_{m,b,k} + 2 \ , \ m = 1, 2, 3\} \ .$$

One or two extra lines of fictitious points are added for convenience in discretizing to second or fourth-order. Boundaries of the computational domain will coincide with the boundaries of the unit cubes, $i_m = n_{m,a,k}$ or $i_m = n_{m,b,k}$ for $m = 1, \dots, n_d$. Henceforth, the subscript k, denoting the component grid, will normally not be written.

Let $\mathbf{U}_i = [R_i, \mathbf{V}_i, Q_i]^T$ denote the discrete approximations to $\mathbf{u} = [\rho, \mathbf{v}, T]^T$

$$\mathbf{U}_i \approx \mathbf{u}(\mathbf{x}_i)$$

The Navier-Stokes equations are discretized with second or fourth-order accurate central differences applied to the equations written in the unit cube coordinates, as will now be outlined. Define the shift operator in the coordinate direction m by

$$E_{+m}\mathbf{U}_i = \begin{cases} \mathbf{U}_{i_1+1,i_2,i_3} & \text{if } m = 1 \\ \mathbf{U}_{i_1,i_2+1,i_3} & \text{if } m = 2 \\ \mathbf{U}_{i_1,i_2,i_3+1} & \text{if } m = 3 \end{cases} , \tag{2}$$

and the difference operators

$$\begin{aligned} D_{+m} &= E_{+m} - 1 \\ D_{+m_1,m_2} &= E_{+m_1}E_{+m_2} - 1 \, . \end{aligned}$$

Let D_{4r_m}, $D_{4r_m r_n}$, D_{4x_m} and $D_{4x_m x_n}$ denote fourth order accurate derivatives with respect to $\mathbf{r}$ and $\mathbf{x}$. The derivatives with respect to $\mathbf{r}$ are the standard fourth-order centred difference approximations. For example

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial r_m} \approx D_{4r_m}\mathbf{U}_i &:= \frac{(-E_{+m}^2 + 8E_{+m} - 8E_{+m}^{-1} + E_{+m}^{-2})\mathbf{U}_i}{12(\Delta r_m)} \\ \frac{\partial^2 \mathbf{u}}{\partial r_m^2} \approx D_{4r_m r_m}\mathbf{U}_i &:= \frac{(-E_{+m}^2 + 16E_{+m} - 30 + 16E_{+m}^{-1} - E_{+m}^{-2})\mathbf{U}_i}{24(\Delta r_m)^2} \end{aligned}$$

where $\Delta r_m = 1/(n_{m,b} - n_{m,a})$. The derivatives with respect to $\mathbf{x}$ are defined by the chain rule.

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial x_m} &= \sum_n \frac{\partial r_n}{\partial x_m}\frac{\partial \mathbf{u}}{\partial r_n} \quad \approx D_{4x_m}\mathbf{U}_i := \sum_n \frac{\partial r_n}{\partial x_m} D_{4r_n}\mathbf{U}_i \\ \frac{\partial^2 \mathbf{u}}{\partial x_m^2} &= \sum_{n,l} \frac{\partial r_n}{\partial x_m}\frac{\partial r_l}{\partial x_m}\frac{\partial^2 \mathbf{u}}{\partial r_n r_l} + \sum_n \frac{\partial^2 r_n}{\partial x_m^2}\frac{\partial \mathbf{u}}{\partial r_n} \\ &\approx D_{4x_m x_m}\mathbf{U}_i := \sum_{n,l} \frac{\partial r_n}{\partial x_m}\frac{\partial r_l}{\partial x_m} D_{4r_n r_l}\mathbf{U}_i + \sum_n \left(D_{4x_m}\frac{\partial r_n}{\partial x_m}\right) D_{4r_n}\mathbf{U}_i \end{aligned}$$

The entries in the Jacobian matrix, $\partial r_m/\partial x_n$, are assumed to be known at the vertices of the grid. Second-order accurate approximations are defined in a similar manner.

The spatial discretization of the equations can thus be written as

$$\frac{d}{dt}R_i + (\mathbf{V}_i \cdot \nabla_4)R_i + R_i(\nabla_4 \cdot \mathbf{V}_i) - \nu_\rho \Delta_4 R_i - \epsilon_\rho \sum_m (D_{+m}D_{-m})R_i$$

$$\frac{d}{dt}\mathbf{V}_i + (\mathbf{V}_i \cdot \nabla_4)\mathbf{V}_i + R_g \frac{Q_i}{R_i}\nabla_4 R_i + R_g \nabla_4 Q_i - \frac{\mu}{R_i}(\Delta_4 \mathbf{V}_i + \dots) = 0$$

$$\frac{d}{dt}Q_i + (\mathbf{V}_i \cdot \nabla_4)Q_i + (\gamma - 1)Q_i(\nabla_4 \cdot \mathbf{V}_i) - (\gamma - 1)\frac{\tilde{k}}{R_i}\Delta_4 Q_i - (\gamma - 1)\frac{\mu}{R_g R_i}\Phi = 0$$

where

$$\begin{aligned}\nabla_4 U_i &= (D_{4x_1}U_i, D_{4x_2}U_i, D_{4x_3}U_i) \\ \nabla_4 \cdot \mathbf{U}_i &= D_{4x_1}U_{1,i} + D_{4x_2}U_{2,i} + D_{4x_3}U_{3,i} \\ \Delta_4 \mathbf{U}_i &= (D_{4x_1x_1} + D_{4x_2x_2} + D_{4x_3x_3})\mathbf{U}_i \ .\end{aligned}$$

We have allowed for two types of “artificial” viscosity in the in the continuity equation for ρ, these are the terms multiplied by ν_ρ and ϵ_ρ.

4.5.2 Computational variables

The computational variables are

$$\mathbf{u} = \begin{bmatrix} \rho \\ \mathbf{v} \\ T \end{bmatrix}$$

in the continuous case and

$$\mathbf{U}_i = \begin{bmatrix} R_i \\ \mathbf{V}_i \\ Q_i \end{bmatrix}$$

in the discrete case. Thus when assigning initial conditions, for example, the first component in the fortran array is the density and the last component is the temperature.

4.6 Method ASF : All Speed Flow algorithm for the compressible Navier-Stokes equations at low Mach Number

When the mach number is low we solve the Navier-Stokes equations in the following form

$$\begin{aligned}
\rho_t + \nabla\cdot(\rho\mathbf{u}) &= 0 \\
\mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \frac{1}{\rho}\nabla p &= \frac{1}{\rho}\mu[\Delta\mathbf{u} + \nabla(\nabla\cdot\mathbf{u})] \\
T_t + (\mathbf{u}\cdot\nabla)T + (\gamma-1)T\nabla\cdot\mathbf{u} &= \frac{1}{\rho c_v}\left\{\nabla\cdot(\lambda\nabla T) + \Phi\right\} \\
&= \frac{\gamma-1}{\rho R}\left\{\nabla\cdot(k\nabla T) + \Phi\right\} p_t + (\mathbf{u}\cdot\nabla)p + \gamma p\nabla\cdot\mathbf{u} \quad = (\gamma-1)\left\{\nabla\cdot(k\nabla T) + \Phi\right\}
\end{aligned}$$

4.7 Compressible Navier-Stokes Equations with Chemistry

The equations we solve are (in general) the compressible Navier-Stokes equations for a chemically reacting flow. Non-reacting flow is of course a special case that can be handled with no loss in efficiency. In non-conservative form the equations are

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = 0 \tag{3}$$

$$\frac{\partial\mathbf{v}}{\partial t} + (\mathbf{v}\cdot\nabla)\mathbf{v} + \frac{1}{\rho}\nabla p = \frac{1}{\rho}\nabla\cdot\boldsymbol{\tau} \tag{4}$$

$$= \frac{\mu}{\rho}[\Delta\mathbf{u} + \frac{1}{3}\nabla(\nabla\cdot\mathbf{u})] \tag{5}$$

$$\frac{\partial e}{\partial t} + (\mathbf{v}\cdot\nabla)e + \frac{p}{\rho}\nabla\cdot\mathbf{v} = -\frac{1}{\rho}\nabla\cdot\mathbf{q} + \frac{1}{\rho}\Phi \tag{6}$$

$$\frac{\partial Y_i}{\partial t} + (\mathbf{v}\cdot\nabla)Y_i = \frac{\sigma_i}{\rho} + \frac{1}{\rho}\nabla\cdot(\rho D_i\nabla Y_i) \qquad i = 1,2,\ldots,n \tag{7}$$

where

$$p = \rho R_u T \sum_{i=1}^{n} \frac{Y_i}{\bar{m}_i} \qquad = \rho R T, \qquad R_i = \frac{R_u}{\bar{m}_i} \tag{8}$$

$$R = \sum_{i=1}^{n} R_i Y_i = R_u \sum_{i=1}^{n} \frac{Y_i}{\bar{m}_i} = \frac{R_u}{\bar{m}}, \qquad \bar{m} = \sum_{i=1}^{n} \bar{m}_i X_i \tag{9}$$

$$e = \sum_{i=1}^{n} (h_i Y_i) - \frac{p}{\rho} \qquad = h - \frac{p}{\rho} \qquad = \sum_{i=1}^{n} (h_i - R_i T) Y_i \tag{10}$$

$$\tau = \mu[\nabla \mathbf{v} + (\nabla \mathbf{v})^T] + (\kappa - \frac{2}{3}\mu)(\nabla \cdot \mathbf{v})\mathbf{I}, \qquad \Phi = \tau : \nabla \mathbf{v} \tag{11}$$

$$h_i(T) = \Delta h_i^0 + \int_{T^0}^{T} c_{p,i}(T) dT \qquad i = 1, 2, ..., n \tag{12}$$

$$\mathbf{q} = -\lambda \nabla T - \rho \sum_{i=1}^{n} h_i D_i \nabla Y_i \tag{13}$$

$$\mathbf{V}_i = -D_i \nabla ln(Y_i) \qquad \text{(assumption)} \tag{14}$$

$$c_p = \sum_{i=1}^{n} c_{p,i} Y_i, \qquad c_v = c_p - R = \sum_{i=1}^{n} (c_{p,i} - R_i) Y_i, \qquad \gamma = \gamma(T, Y_i) = \frac{c_p}{c_v} \tag{15}$$

Depending on the flow regime (primarily Mach number) we may also want to use the temperature equation

$$\frac{DT}{Dt} + (\gamma - 1) T \nabla \cdot \mathbf{v} = \frac{1}{\rho c_v} \left\{ \sum_{i=1}^{n} (R_i T - h_i) \sigma_i + \nabla \cdot (\lambda \nabla T) + \sum_{i=1}^{n} R_i T \nabla \cdot (\rho D_i \nabla Y_i) + \rho \sum_{i=1}^{n} c_{p,i} D_i \nabla T \cdot \nabla Y_i + \right. \tag{16}$$

and/or the pressure equation

$$\frac{Dp}{Dt} + \gamma p \nabla \cdot \mathbf{v} = \sum_{i=1}^{n} (\gamma R_i T - (\gamma - 1) h_i) \sigma_i + (\gamma - 1) \nabla \cdot (\lambda \nabla T) \tag{17}$$

$$+ \sum_{i=1}^{n} \gamma R_i T \nabla \cdot (\rho D_i \nabla Y_i) + (\gamma - 1) \rho \sum_{i=1}^{n} c_{p,i} D_i \nabla T \cdot \nabla Y_i + (\gamma - 1) \Phi \tag{18}$$

See Bill's combustion notes for derivation of these equations.

4.8 Axisymmetric Problems

Here I describe the equations that are solved for axisymmetric problems.

Let the cylindrical coordinates be (x, r, θ) where x is the axial variable, r the radial variable and θ is the azimuthal angle about the axis $r = 0$. Let the velocity be $\mathbf{u} = U\hat{\mathbf{x}} + V\hat{\mathbf{r}} + W\hat{\boldsymbol{\theta}}$ where (U, V, W) are the components of axial, radial and azimuthal velocities and $(\hat{\mathbf{x}}, \hat{\mathbf{r}}, \hat{\boldsymbol{\theta}})$ are the three unit vectors in the coordinate directions.

In cylindrical coordinates we have the general relations

$$\begin{aligned}
\mathbf{F} &= \hat{\mathbf{x}}F^x + \hat{\mathbf{r}}F^r + \hat{\boldsymbol{\theta}}F^\theta \\
\nabla V &= \hat{\mathbf{x}}V_x + \hat{\mathbf{r}}V_r + \frac{\hat{\boldsymbol{\theta}}}{r}V_\theta \\
\mathbf{n}\cdot\nabla\mathbf{F} &= \hat{\mathbf{x}}(\mathbf{n}\cdot\nabla F^x) + \hat{\mathbf{r}}(\mathbf{n}\cdot\nabla F^r - \frac{n^\theta F^\theta}{r}) + \hat{\boldsymbol{\theta}}(\mathbf{n}\cdot\nabla F^\theta + \frac{n^\theta F^r}{r}) \\
\nabla\cdot\mathbf{F} &= F^x_x + \frac{1}{r}(rF^r)_r + \frac{1}{r}F^\theta_\theta \\
\Delta V &= V_{xx} + \frac{1}{r}(rV_r)_r + \frac{1}{r^2}V_{\theta\theta} \\
\Delta\mathbf{F} &= \hat{\mathbf{x}}(\Delta F^x) + \hat{\mathbf{r}}(\Delta F^r - \frac{F^r}{r^2} - \frac{2}{r^2}F^\theta_\theta) + \hat{\boldsymbol{\theta}}(\Delta F^\theta + \frac{2}{r^2}F^r_\theta - \frac{F^\theta}{r^2}) \\
\nabla\times\mathbf{F} &= \hat{\mathbf{x}}(\frac{1}{r}(rF^\theta)_r - \frac{1}{r}F^r_\theta) + \hat{\mathbf{r}}(\frac{1}{r}F^x_\theta - F^\theta_x) + \hat{\boldsymbol{\theta}}(F^r_x - F^x_r)
\end{aligned}$$

The incompressible Navier-Stokes equations in cylindrical coordinates are (see Batchelor)

$$\begin{aligned}
U_t + UU_x + VU_r + \frac{W}{r}U_\theta + p_x &= \nu(U_{xx} + \frac{1}{r}(rU_r)_r + \frac{1}{r^2}U_{\theta\theta}) \\
V_t + UV_x + VV_r + \frac{W}{r}V_\theta - \frac{W^2}{r}p_r &= \nu(V_{xx} + \frac{1}{r}(rV_r)_r + \frac{1}{r^2}V_{\theta\theta} - \frac{V}{r^2} - \frac{2}{r^2}W_\theta) \\
W_t + UW_x + VW_r + \frac{W}{r}W_\theta + \frac{VW}{r} + \frac{1}{r}p_\theta &= \nu(W_{xx} + \frac{1}{r}(rW_r)_r + \frac{1}{r^2}W_{\theta\theta} - \frac{W}{r^2} + \frac{2}{r^2}V_\theta) \\
U_x + \frac{1}{r}(rV)_r + \frac{1}{r}W_{\theta\theta} &= 0
\end{aligned}$$

For axisymmetric problems with no swirl, $W = 0$ and all derivatives with respect to θ are zero,

$$\begin{aligned}
U_t + UU_x + VU_r + p_x &= \nu(U_{xx} + \frac{1}{r}(rU_r)_r \\
V_t + UV_x + VV_r + p_r &= \nu(V_{xx} + \frac{1}{r}(rV_r)_r - \frac{V}{r^2}) \\
U_x + \frac{1}{r}(rV)_r &= 0
\end{aligned}$$

The divergence of the advection terms is

$$\begin{aligned}
\nabla\cdot(UU_x + VU_r, UV_x + VV_r) &= (UU_x + VU_r)_x + \frac{1}{r}[r(UV_x + VV_r)]_r \\
&= U_x^2 + 2V_xU_r + V_r^2 + U\{(U_x)_x + \frac{1}{r}[r(V_x)_r]\} + V\{(U_r)_x + \frac{1}{r}[r(V_r)]_r\} \\
&= U_x^2 + 2V_xU_r + V_r^2 + U(\nabla\cdot\mathbf{U})_x + V(\nabla\cdot\mathbf{U})_r
\end{aligned}$$

and thus pressure equation becomes

$$p_{xx} + \frac{1}{r}(rp_r)_r = U_x^2 + 2V_xU_r + V_r^2$$

The boundary conditions on the axis of symmetry are

$$\begin{aligned}
U_r(x,0) &= 0 \\
V(x,0) &= 0 \quad , \quad V_{rr}(x,0) = 0 \\
P_r(x,0) &= 0
\end{aligned}$$

In general all odd derivatives of $\mathbf{U}, p$ with respect to r will be zero at $r = 0$.

Note that for r small,

$$\begin{aligned}
\frac{1}{r}(rV_r)_r - \frac{V}{r^2} &= V_{rr} + \frac{1}{r}V_r - \frac{V}{r^2} \\
&= V_{rr} + \frac{1}{r}(V_r(x,0) + rV_{rr}(x,0) + O(r^2)) - \frac{1}{r^2}(V(x,0) + rV_r(x,0) + \frac{R^2}{2}V_{rr}(x,0) + O(r^3) \\
&= \frac{3}{2}V_{rr}(x,0) + O(r) \\
&= O(r)
\end{aligned}$$

and

$$\frac{1}{r}(rU_r)_r = U_{rr} + \frac{1}{r}U_r = 2U_{rr} + O(r)$$

We can use these last two results to evaluate the viscous terms on the boundary $r = 0$, (eliminating the removable singularity) althought the dirichlet condition $V(x,0) = 0$ obviates the need for the later equation on the boundary.

Note that in inviscid flow the only difference between the axi-symmetric equations and the 2D equations is a change to the pressure equation (or to the incompressibility equation) with the addition of the $\frac{1}{r}p_r$ term. With viscosity there are also differences in the viscous terms.

5 Motion of rigid bodies

The class `RigidBodyMotion` can be used to track the motion of a rigid body moving under the influence of forces and torques.

A `RigidBodyMotion` object must be initialized with the basic information about a body such as the mass, moments of inertia and axes of inertia, in addition to the initial position and velocities.

A rigid body moves under the influence of a force, $\mathbf{F}(t)$ and torque $\mathbf{G}(t)$. These forces and torques should be supplied at a sequence of times, $(t_i, \mathbf{F}(t_i), \mathbf{G}(t_i))$. The rigid body object will integrate the equations of motion and supply the currect position and orientation.

The equations of motion for a rigid body in the standard cartesian reference frame are

$$\begin{aligned} \frac{d\mathbf{x}}{dt} &= \mathbf{v} \\ M\frac{d\mathbf{v}}{dt} &= \mathbf{F} \\ \frac{d\mathbf{h}}{dt} &= \mathbf{G} \end{aligned}$$

where

M	:	mass of the body
$\mathbf{x}$	:	position of the centre of mass
$\mathbf{v}$	:	velocity of the centre of mass
$\mathbf{h}$	:	angular momentum
$\mathbf{F}$	:	resultant force
$\mathbf{G}$	:	resultant torque about the centre of mass

The torque $\mathbf{G}$ will usually be defined as

$$\mathbf{G} = \int_{\partial B} (\mathbf{r} - \mathbf{x}_{\text{cm}}) \times d\mathbf{F} \qquad \text{torque on the body}$$

where the integral is over the surface of the rigid body, ∂B.

It is convenient to represent the angular momentum, $\mathbf{h}$, in terms of the principle moments of inertia, I_i, the principle axes of inertia, $\mathbf{e}_i$, and the angular velocities ω_i about each axis $\mathbf{e}_i$

$$\begin{aligned} \mathbf{h} &= \sum I_i \omega_i \mathbf{e}_i \\ \mathbf{e}_i \cdot \mathbf{e}_j &= \delta_{ij} \end{aligned}$$

Since the principle axes of inertia remain an orthonormal basis it follows that

$$\dot{\mathbf{e}}_i = \boldsymbol{\omega} \times \mathbf{e}_i \qquad (\mathbf{e}_i \cdot \mathbf{e}_i = 1,\ \ \mathbf{e}_i \cdot \dot{\mathbf{e}}_i = 0)$$

and then the equation for the angular momentum becomes

$$\begin{aligned} I_i \dot{\omega}_i &= \mathbf{G} \cdot \mathbf{e}_i \\ \dot{\mathbf{e}}_i &= \boldsymbol{\omega} \times \mathbf{e}_i \qquad (\mathbf{e}_i \cdot \mathbf{e}_i = 1,\ \ \mathbf{e}_i \cdot \dot{\mathbf{e}}_i = 0) \end{aligned}$$

where

$$\begin{aligned} I_i &: \text{ principle moments of inertia} \\ \mathbf{e}_i &: \text{ principle axes of inertia} \\ \boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3) & \text{angular velocities about } \mathbf{e}_i \end{aligned}$$

In summary we solve the set of ODEs

$$M\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F} \tag{19}$$

$$I_i\dot{\omega}_i = \mathbf{G}\cdot\mathbf{e}_i \tag{20}$$

$$\dot{\mathbf{e}}_i = \boldsymbol{\omega}\times\mathbf{e}_i \qquad (\mathbf{e}_i\cdot\mathbf{e}_i = 1,\;\; \mathbf{e}_i\cdot\dot{\mathbf{e}}_i = 0) \tag{21}$$

We integrate the motion of the principle axes, $\mathbf{e}_i(t)$ over time along with the position of the center of mass.

The rotation matrix that must be applied to rotate the body from it's position at $t = 0$ to any time t is simply $E(t)E^{-1}(0)$ where E is the matrix with columns being $\mathbf{e}_i$,

$$R(t) = E(t)E^{-1}(0) = \begin{bmatrix}\mathbf{e}_0(t) & \mathbf{e}_1(t) & \mathbf{e}_2(t)\end{bmatrix}\begin{bmatrix}\mathbf{e}_0^T(0) \\ \mathbf{e}_1^T(0) \\ \mathbf{e}_2^T(0)\end{bmatrix}$$

Thus the position, $\mathbf{r}(t)$, of a point $\mathbf{r}$ on the surface of the body will be given by the sum of the position of the centre of mass, $\mathbf{x}(t)$ plus a rotation.

$$\mathbf{r}(t) = \mathbf{x}(t) + R(t)(\mathbf{r}(0) - \mathbf{x}(0))$$

Whence the velocity and acceleration of the point are

$$\begin{aligned} \dot{\mathbf{r}}(t) &= \mathbf{v}(t) + \dot{R}(\mathbf{r}(0) - \mathbf{x}(0)) \\ &= \mathbf{v}(t) + \sum_i(\boldsymbol{\omega}\times\mathbf{e}_i)(r_i(0) - x_i(0)) \\ \ddot{\mathbf{r}}(t) &= \mathbf{F}/M + \ddot{R}(\mathbf{r}(0) - \mathbf{x}(0)) \\ &= \mathbf{F}/M + \sum_i(\dot{\boldsymbol{\omega}}\times\mathbf{e}_i + \boldsymbol{\omega}\times\dot{\mathbf{e}}_i)(r_i(0) - x_i(0)) \\ &= \mathbf{F}/M + \sum_i(\dot{\boldsymbol{\omega}}\times\mathbf{e}_i + \boldsymbol{\omega}\times(\boldsymbol{\omega}\times\mathbf{e}_i)(r_i(0) - x_i(0)) \\ &= \mathbf{F}/M + \sum_i(\dot{\boldsymbol{\omega}}\times\mathbf{e}_i + (\boldsymbol{\omega}\cdot\mathbf{e}_i)\boldsymbol{\omega} + |\boldsymbol{\omega}|^2\mathbf{e}_i)(r_i(0) - x_i(0)) \end{aligned}$$

5.1 Background

Reference *Fundamentals of Mechanics* by Synge and Griffith.

The angular momentum $\mathbf{h}$ of a rigid body about it's centre of mass is

$$\mathbf{h} = \int_B(\mathbf{r} - \mathbf{x}_{\text{cm}})\times d\mathbf{p}$$

where $d\mathbf{p}$ is the momentum of a volume element and where the integral is over the volume occupied by the body.

Given a rigid body we can define the moment of inertial about a line $\mathbf{l}$ to be

$$\begin{aligned} I &= \int m(\mathbf{x}) r(\mathbf{x})^2 d\mathbf{x} \\ r &= |\hat{\mathbf{l}} \times \mathbf{x}| \end{aligned}$$

where $\hat{\mathbf{l}}$ is a unit vector in the direction of $\mathbf{l}$ and thus r is the distance from a point $\mathbf{x}$ in the body to the line $\mathbf{l}$.

We can define the principle moments of inertia and the principle axes of inertia as follows. Define

$$\begin{aligned} A_{11} &= \int_B m(x_2^2 + x_3^2) d\mathbf{x} \\ A_{22} &= \int_B m(x_3^2 + x_1^2) d\mathbf{x} \\ A_{33} &= \int_B m(x_1^2 + x_2^2) d\mathbf{x} \\ A_{13} &= -\int_B m x_3 x_1 d\mathbf{x} \\ A_{12} &= -\int_B m x_1 x_2 d\mathbf{x} \\ A_{23} &= -\int_B m x_2 x_3 d\mathbf{x} \end{aligned}$$

The principle moments of inertia are the eigenvalues, I_i, of

$$A\mathbf{e} = I\mathbf{e}$$

while the eigenvectors, $\mathbf{e}_i$, are the principle axes of inertia.

Let $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)$ be the angular velocities of the body about the principle axes of inertia. Let $\boldsymbol{\Omega}$ be the angular velocity of the principle axes, $\mathbf{e}_i$ (normally $\boldsymbol{\Omega} = \boldsymbol{\omega}$ but in symmetric problems we may not need to rotate the some of the principle axes).

The equations of motion of the rigid body in the (rotating) reference frame attached to the principle axes $\mathbf{e}_i$ are

$$\begin{aligned} M\frac{d\mathbf{v}}{dt} &= \frac{\partial \mathbf{v}}{\partial t} + \boldsymbol{\Omega} \times \mathbf{v} = \mathbf{F} \\ \frac{d\mathbf{h}}{dt} &= \frac{\partial \mathbf{h}}{\partial t} + \boldsymbol{\Omega} \times \mathbf{h} = \mathbf{G} \end{aligned}$$

where

$$\begin{aligned} M &= \text{total mass of the body} \\ \mathbf{x}_{\text{cm}} &= \text{center of mass} \\ \mathbf{v} &= \text{velocity of the center of mass} \\ \mathbf{h} &= \sum I_i \omega_i \mathbf{e}_i \qquad \text{angular momentum} \\ \mathbf{F} &= \text{ external force on the body} \\ \mathbf{G} &= \int (\mathbf{r} - \mathbf{x}_{\text{cm}}) \times d\mathbf{F} \qquad \text{torque on the body} \end{aligned}$$

Written out in components, the equations for the 6 degrees of freedom of a rigid body are

$$\begin{aligned}
M(\dot{v}_1 - v_2\Omega_3 + v_3\Omega_2) &= F_1 \\
M(\dot{v}_2 - v_3\Omega_1 + v_1\Omega_3) &= F_2 \\
M(\dot{v}_3 - v_1\Omega_2 + v_2\Omega_1) &= F_3 \\
I_1\dot{\omega}_1 - I_2\omega_2\Omega_3 + I_3\omega_3\Omega_2 &= G_1 \\
I_2\dot{\omega}_2 - I_3\omega_3\Omega_1 + I_1\omega_1\Omega_3 &= G_2 \\
I_3\dot{\omega}_3 - I_1\omega_1\Omega_2 + I_2\omega_2\Omega_1 &= G_3
\end{aligned}$$

These equations need to be integrated to determine the coordinates of the centre of mass, $\mathbf{x}_{\rm cm}(t)$, and the angles of rotation $\boldsymbol{\theta}(t)$ ($\dot{\boldsymbol{\theta}} = \boldsymbol{\omega}$) about the principle axes.

6 OB_Parameters class

6.1 Variables in OB_Parameters

int numberOfDimensions: number of spacial dimensions.

PDE pde: one of

int numberOfComponents: number of components in the equations.

real cfl, cflMin, cflOpt, cflMax: parameters to determine the time step.

int rc: if rc>0 then the density is `u(all,all,all,rc)`.

int uc: if uc>0 then the x component of the velocity is `u(all,all,all,uc)`.

int vc: if vc>0 then the y component of the velocity is `u(all,all,all,vc)`.

int wc: if wc>0 then the z component of the velocity is `u(all,all,all,wc)`.

int pc: if pc>0 then the pressure is `u(all,all,all,pc)`.

int tc: temperature

int sc: position of first species, species m is located at sc+m

int kc, epsc: for k-epsilon model

real machNumber, reynoldsNumber, prandtlNumber: PDE parameters CNS and ASF

real mu, kThermal, Rg, gamma, avr, anu: for CNS, ASF

real pressureLevel, nuRho: for ASF

enum TurbulenceModel turbulenceModel: One of

```
enum TurbulenceModel
{
  noTurbulenceModel,
  BaldwinLomax,
  kEpsilon,
  kOmega,
  SpalartAllmaras,
  numberOfTurbulenceModels
};
```

Boundary condition parameters:

IntegerArray bcInfo(0: 2,side,axis,grid): Array holding info about the parameters. The values are accessed through member functions `bcType(side,axis,grid)`, `variableBoundaryData(side,axis,grid)` and `bcIsTimeDependent(side,axis,grid)`

bcInfo(0,side,axis,grid) : values from enum BoundaryConditionType, e.g. uniformInflow

bcInfo(1,side,axis,grid) : bit flag, bit 1=BC is spatially dependent, bit 2=BC is time dependent.

RealArray bcData: bcData(.,side,axis,grid) : data for the boundary condition

real inflowPressure:

RealArray bcParameters: arrays for boundary condition parameters

IntegerArray variableBoundaryData: variableBoundaryData(grid) is true if variable BC data is required.

6.2 Constructor

OB_Parameters(const int & numberOfDimensions0, const PDE & pde0)

pde0: Indicated which PDE we are solving

6.3 buildErrorEstimator

int
buildErrorEstimator()

Description:

6.4 updateToMatchGrid

int
updateToMatchGrid(CompositeGrid & cg,
IntegerArray & sharedBoundaryCondition = Overture::nullIntArray())

Description: Update the parameters when the grid has changed.

sharedBoundaryCondition(side,axis,grid) : = side2+2*(axis2+3*grid2) : match to (side2,axis2,grid2)

6.5 bcType

int
bcType(int side, int axis, int grid) const

Description: Return the boundary condition type, a value from the enum `BoundaryConditionType`:

```
enum BoundaryConditionType
{
  uniformInflow,
  parabolicInflow,
  rampInflow,
  userDefinedBoundaryData
};
```

side,axis,grid (input): indicates a face of a grid.

Return value: the boundary condition type for a given face of a grid.

6.6 howManyBcTypes

int
howManyBcTypes(const Index & side,
const Index & axis,
const Index & grid,
BoundaryConditionType bc) const

Description: Return the number of faces where there is a boundary condition of type ”bc”, from the specified faces.

side,axis,grid: check these faces.

bc (input): check for this boundary condition.

Return value: number of faces where the boundary condition is ”bc”

6.7 howManyTimeDependentUserBoundaryConditions

int
thereAreTimeDependentUserBoundaryConditions(const Index & Side,
const Index & Axis,
const Index & Grid) const

Description: Return the number of faces where there is a time dependent user boundary condition.

Side,Axis,Grid: check these faces.

Return value: number of faces where the boundary condition is ”bc”

6.8 setBcType

int
setBcType(int side, int axis, int grid, BoundaryConditionType bc)

6.9 bcType

int
userBcType(int side, int axis, int grid) const

Description: Return the user defined boundary condition type.

side,axis,grid (input): indicates a face of a grid.

Return value: the boundary condition type for a given face of a grid.

6.10 bcVariesInSpace

int
bcVariesInSpace(int side, int axis, int grid) const

6.11 bcVariesInSpace

int
bcVariesInSpace(const Index & side = nullIndex,
const Index & axis = nullIndex,
const Index & grid = nullIndex) const

6.12 setBcVariesInSpace

int
setBcVariesInSpace(int side, int axis, int grid, bool trueOrFalse = true)

6.13 bcIsTimeDependent

int
bcIsTimeDependent(int side, int axis, int grid) const

6.14 setBcIsTimeDependent

int
setBcIsTimeDependent(int side, int axis, int grid, bool trueOrFalse = true)

6.15 setUserBoundaryConditionParameters

int
setUserBoundaryConditionParameters(int side, int axis, int grid, RealArray & values)

6.16 getUserBoundaryConditionParameters

int
getUserBoundaryConditionParameters(int side, int axis, int grid, RealArray & values) const

values (input/output) : on input must be the correct length.

6.17 setTimeDependenceBoundaryConditionParameters

int
setTimeDependenceBoundaryConditionParameters(int side, int axis, int grid, RealArray & values)

6.18 getTimeDependenceBoundaryConditionParameters

int
getTimeDependenceBoundaryConditionParameters(int side, int axis, int grid, RealArray & values) const

values (input/output) : on input must be the correct length.

6.19 addShowVariable

int
addShowVariable(const aString & name, int component, bool variableIsOn = TRUE)

Description: Add a show variable name to the list of possible show file variables.

name (input) : name to give the show variable

component (input) : the component number of this variable (if it is a computational variable), otherwise a positive integer larger than any component number.

variableIsOn (input) : if true this variable will be saved in the show file. If false the variable will not be saved by default but the user can change this.

Notes: showVariable(i) >0 if we are to save showVariableName[i], < 0 we do not save. showVariable-Name[] names of possible variables to save in the show file, NULL terminated.

6.20 getGridIsImplicit

int
getGridIsImplicit(int grid) const

Description: Return 1 or 2 if the grid is integrated implicitity. This requires that both the time stepping method is an implicit one and that the grid was chosen to be implicit.

Return value: 1=implicit, 2= semi-implicit

6.21 setGridIsImplicit

int
setGridIsImplicit(int grid =-1 */, int value /* =1)

Description: Specify if this grid should be integrated implicitly when an implicit time stepping method is used.

grid (input) : grid to set, -1=set all

value (input) : 1=implicit, 2=semi-implicit, 0 = not-implicit

6.22 useConservativeVariables

bool
useConservativeVariables() const

Description: if true we are using a solver that uses conservative variables

6.23 isAxisymmetric

bool
isAxisymmetric() const

Description: If true we are solving an axisymmetric problem on a 2D grid

6.24 isSteadyStateSolver

bool
isSteadyStateSolver() const

Description: If true we are solving a steady state problem.

6.25 setTwilightZoneFunction

int
setTwilightZoneFunction(const TwilightZoneChoice & choice,
const int & degreeSpace =2,
const int & degreeTime =1)

Description:

choice (input): OB_Parameters::polynomial or OB_Parameters::trigonometric

6.26 updateShowFile

int
updateShowFile(const aString & command = nullString,
DialogData *interface =NULL)

Description: Open or close show files, set variables that appear in the show file.

command (input) : optionally supply a command to execute. Attempt to execute the command and then return. The return value is 0 if the command was executed, 1 otherwise.

interface (input) : use this dialog. If command==”build dialog”, fill in the dialog and return.

Return value: when executing a single command, return 1 if the command was not recognised.

Here is a desciption of the menu options available for changing show file options.

open : open a new show file.

close : close any open show file.

show file variables : specify extra derived quantities, such as the divergence or vorticity, that should be saved in the show file in addition to the standard variables.

frequency to save : By default the solution is saved in the show file as often as it is plotted according to `'times to plot'`. To save the solution less often set this integer value to be greater than 1. A value of 2 for example will save solutions every 2nd time the solution is plot.

frequency to flush : Save this many solutions in each show file so that multiple show files will be created (these are automatically handled by plotStuff). See section (**??**) for why you might do this.

properties :

uncompressed : save the show file uncompressed. This is a more portable format that can be read by newer versions of Overture.

compressed : save the show file compressed. This is a less portable format.

7 Notes

Here is where Bill keeps some notes for himself. These are not meant to be comprehensible to anyone else.

7.1 Slip wall versus symmetry wall

The slip wall BC for INS imposes $\nabla \cdot \mathbf{u} = 0$ – for comparing an extruded 3d run to a 2D run it better to use the symmetry BC since the divergence BC may generate a non-constant w.

7.2 Moving Grids

On a non-moving grid, each component grid is defined by a mapping of the form

$$\mathbf{x} = \mathbf{G}(\mathbf{r})$$

where $\mathbf{r}$ denotes the coordinates on the unit-square (or unit-cube).

On a moving grid the mapping function depends on time:

$$\mathbf{x} = \mathbf{G}(\mathbf{r}, t)$$

For example, a rotating square is defined by the mapping

$$\begin{aligned} \mathbf{x} = \mathbf{G}(\mathbf{r}, t) &= R(t)\mathbf{r} \\ R(t) &= \begin{bmatrix} \cos(\omega t) & -\sin(\omega t) \\ \sin(\omega t) & \cos(\omega t) \end{bmatrix} \end{aligned}$$

On a moving grid we solve the PDE in a frame that moves with the grid. Thus if we were solving

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla_{\mathbf{x}})\mathbf{u} + \nabla_{\mathbf{x}} p = \nu \Delta_{\mathbf{x}} \mathbf{u}$$

then on each grid we make the change of variables from $(\mathbf{x}, t)$ to $(\mathbf{r}, \tau)$ defined by

$$\begin{aligned} \mathbf{x} &= \mathbf{G}(\mathbf{r}, \tau) \\ t &= \tau \\ \mathbf{u}(\mathbf{x}, t) &= \mathbf{u}(\mathbf{G}(\mathbf{r}, \tau), \tau) \equiv \mathbf{U}(\mathbf{r}, \tau) \end{aligned}$$

Whence

$$\begin{aligned} \frac{\partial}{\partial x_i}\mathbf{u}(\mathbf{x}, t) &= \frac{\partial r_j}{\partial x_i}\frac{\partial}{\partial r_j}\mathbf{U}(\mathbf{r}, \tau) + \frac{\partial \tau}{\partial x_i}\frac{\partial}{\partial \tau}\mathbf{U}(\mathbf{r}, \tau) \qquad \text{(with summation convention)} \\ &= \frac{\partial r_j}{\partial x_i}\frac{\partial}{\partial r_j}\mathbf{U}(\mathbf{r}, \tau) \end{aligned}$$

and

$$\frac{\partial}{\partial \tau}\mathbf{U} = \frac{\partial \mathbf{G}}{\partial \tau}\frac{\partial}{\partial x_i}\mathbf{u}(\mathbf{x}, t) + \frac{\partial t}{\partial \tau}\frac{\partial}{\partial t}\mathbf{u}$$

implies

$$\begin{aligned} \frac{\partial}{\partial t}\mathbf{u} &= \frac{\partial}{\partial \tau}\mathbf{U} - \dot{\mathbf{G}} \cdot \nabla_{\mathbf{x}}\mathbf{u} \\ &= \frac{\partial}{\partial t}\mathbf{U} - \dot{G}_i(\nabla_{\mathbf{x}}\mathbf{r} \cdot \nabla_{\mathbf{r}})\mathbf{U} \end{aligned}$$

where

$$\dot{\mathbf{G}} \equiv \frac{\partial \mathbf{G}(\mathbf{r},\tau)}{\partial \tau}$$

is the grid velocity. The incompressible Navier-Stokes equations

$$\mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \nu \Delta \mathbf{u} \tag{22}$$

$$\Delta p + \sum_i \nabla u_i \cdot \partial_{x_i}\mathbf{u} = 0 \tag{23}$$

transform to

$$\begin{aligned} \mathbf{U}_\tau + [(U_i - \dot{G}_i)(\partial_{x_i} r_j)\partial_{r_j}]\mathbf{U} + (\partial_{x_i} r_j)\partial_{r_j} P &= \nu \tilde{\Delta}\mathbf{U} \\ \tilde{\Delta} P + \sum_i \tilde{\nabla} U_i \cdot \partial_{x_i}\mathbf{U} &= 0 \end{aligned}$$

or with a simplified notation:

$$\mathbf{U}_\tau + [(\mathbf{U} - \dot{\mathbf{G}})\cdot\tilde{\nabla}]\mathbf{U} + \tilde{\nabla}P = \nu\tilde{\Delta}\mathbf{U} \tag{24}$$

In conservative form the equation

$$\rho_t + \nabla_{\mathbf{x}} \cdot (\rho \mathbf{u}) = 0$$

transforms in the moving coordinate system to

$$R_\tau - (\dot{\mathbf{G}}\cdot\nabla)R + \nabla\cdot(R\mathbf{U}) = 0$$

with $\rho(\mathbf{x},t) = R(\mathbf{r},\tau)$. This can be written in a conservation form (with a source term)

$$R_\tau + \nabla\cdot(R(\mathbf{U} - \dot{\mathbf{G}})) + (\nabla\cdot\dot{\mathbf{G}})\, R = 0$$

*** finish this next part: Alternatively we can consider a control volume formulation on a time dependent volume, $V(t)$

$$\partial_t \int_{V(t)} \rho d\mathbf{x} + \int_{V(t)} \nabla_{\mathbf{x}}(\mathbf{F}) d\mathbf{x} = 0$$

or

$$(J(t)\rho)_t + \nabla_{\mathbf{x}}(\mathbf{F})J(t) = 0$$

or

$$\rho_t + \nabla_{\mathbf{x}}(\mathbf{F}) + \frac{1}{J(t)}\partial_t J\, \rho = 0$$

7.2.1 Twilightzone flow

Twilightzone forcing should be thought of as being added to the equations in the non moving coordinate system. This forcing will not change under the change of variables to the moving grid coordinates.

Even the simplest moving grid will cause the twilightzone polynomials to become higher order.

Suppose the twilight zone solution is of the form of a polynomial

$$u(\mathbf{x},t) = X_m(\mathbf{x})T_n(t) \qquad \text{X degree m, T degree n}$$

then

$$U(\mathbf{r},t) = u(G(\mathbf{r},\tau),\tau) = X(G(\mathbf{r},t))T(\tau)$$

If G is a polynomial of degree p then U will be a polynomial of degree $pm+n$. For example if G is a simple translation in the direction $\mathbf{d}$, $G(\mathbf{r},t) = \mathbf{r} + t\mathbf{d}$ ($p=1$) then

$$U(\mathbf{r},t) = X(\mathbf{r}+t\mathbf{d})T(t)$$

and so if u is a linear polynomial

$$u(\mathbf{x},t) = (1+x+y)*(1+t)$$

then

$$U(\mathbf{r},t) = (1+(r_1+td_1)+(r_2+td_2))(1+t)$$

is a quadratic polynomial (in time).
Midpoint rule: exact for $(m,n,p)=(1,0,1)$.
2nd order multi-step rule: exact for $(m,n,p)=(1,1,1)$, or $(m,n,p)=(2,0,1)$.

If G is a rotation then U will no-longer even be a polynomial.

7.2.2 Boundary Conditions

The boundary condition on the velocity at a wall are

$$\begin{aligned} \mathbf{U}(\mathbf{r},t) &= \dot{\mathbf{G}}(\mathbf{r},\tau) \qquad &&\text{: no-slip wall} \\ \mathbf{n}\cdot\mathbf{U} &= \mathbf{n}\cdot\dot{\mathbf{G}}(\mathbf{r},\tau) \qquad &&\text{: slip wall} \end{aligned}$$

On a moving no-slip wall the boundary condition for the pressure equation is obtained by dotting the normal $\mathbf{n}$ into the momentum equation:

$$\begin{aligned} \partial_n P &= -\mathbf{n}\cdot\mathbf{U}_\tau + \nu\mathbf{n}\cdot\tilde{\Delta}\mathbf{U} \\ &= -\mathbf{n}\cdot\ddot{\mathbf{G}} + \nu\mathbf{n}\cdot\tilde{\Delta}\mathbf{U} \end{aligned}$$

Note that the acceleration of the wall appears on the right hand side.

7.2.3 Computing forces on bodies

The force exerted by a fluid on a small surface element immersed in the fluid is

$$dF_i = (pn_i - n_k\tau_{ki})\,dS$$

where $\mathbf{n} = [n_1,n_2,n_3]^T$ is the unit normal to the surface element and $\boldsymbol{\tau}$ is the stress tensor.

$$\boldsymbol{\tau} = \mu[\nabla\mathbf{v} + (\nabla\mathbf{v})^T] + (\kappa - \frac{2}{3}\mu)(\nabla\cdot\mathbf{v})\mathbf{I}$$

Thus the average force applied to a body with boundary $\partial\Omega$ is

$$F_i^{\rm cm} = \int_{\partial\Omega} p n_i - n_k \tau_{ki} dS$$

so that the centre of mass, $\mathbf{x}_{\rm cm}$ will obey Newton's law:

$$M \frac{d^2\mathbf{x}_{\rm cm}}{dt^2} = \mathbf{F}^{\rm cm}$$

where M is the total mass of the body.

7.3 Solving the pressure equation

- The GMRES solver goes haywire if the right-hand side gets large values in it at unused points. One symptom is that it appears to converge in 2 steps. Another is that it blows up later on.
- Extrapolating the pressure in time in AB2 before GMRES helps (2 times fewer iterations ?)
- For moving grids could increase number of corrector steps as this should be inexpensive and allows a bigger time step.

7.4 Turbulence models

7.4.1 LES-SGS

If the flow is sufficiently resolved then one can perform a large-eddy-simulation using a sub-grid-scale model. This basically boils down to using an artificial viscosity of a certain form.

7.5 RANS turbulence models

7.5.1 $k - \epsilon$

We may also need to add a RANS turbulence model such as the $k - \epsilon$ model in which case the viscosity coefficient ν is replaced by $\nu_0(k, \epsilon)$ and ****** these are for incompressible flow only *****

$$\left.\begin{array}{rcl} k_t + (\mathbf{u}\cdot\nabla)k - G + \epsilon - \nabla\cdot(\nu_k \nabla k) &=& 0 \\ \epsilon_t + (\mathbf{u}\cdot\nabla)\epsilon - C_{\epsilon 1}(\epsilon/k)G + C_{\epsilon 2}(\epsilon^2/k) - \nabla\cdot(\nu_\epsilon \nabla\epsilon) &=& 0 \end{array}\right\} \qquad (25)$$

where

ν_T	$= C_\mu(k^2/\epsilon)$	turbulent eddy viscosity
ν_0	$= \nu + \nu_T$	total viscosity
ν_k	$= \alpha_k \nu_0$	viscosity coefficient for k
ν_ϵ	$= \alpha_\epsilon \nu_0$	viscosity coefficient for ϵ
G	$= \nu_T \sum_{ij} \partial_i u_j(\partial_i u_j + \partial_j u_i)$	turbulence generation term

7.5.2 $k-\omega$

The $k-\omega$ turbulence model (for incompressible flow) is

$$\left.\begin{array}{rcl} k_t + (\mathbf{u}\cdot\nabla)k - G + \beta^* k\omega - \nabla\cdot(\nu_k \nabla k) & = & 0 \\ \omega_t + (\mathbf{u}\cdot\nabla)\omega - \alpha\frac{\omega}{k}G - \beta\omega^2 - \nabla\cdot(\nu_\omega \nabla\omega) & = & 0 \end{array}\right\} \tag{26}$$

where

ν_T	$=$	$C_\mu(k^2/\epsilon)$	turbulent eddy viscosity
ν_0	$=$	$\nu + \nu_T$	total viscosity
ν_k	$=$	$\alpha_k \nu_0$	viscosity coefficient for k
ν_ω	$=$	$\alpha_\epsilon \nu_0$	viscosity coefficient for ω
G	$=$	$\nu_T \sum_{ij} \partial_i u_j(\partial_i u_j + \partial_j u_i)$	turbulence generation term

7.5.3 Corner compatibility condition

At a corner between two no-slip walls there is a compatibility condition implied by the pressure boundary condition.

Suppose we have a square grid with a corner at the origin. Then

$$p_x(0,y) = \nu\Delta u - u_t - (uu_x + vu_y)$$
$$p_y(x,0) = \nu\Delta v - v_t - (uv_x + vv_y)$$

Since ...

8 Convergence results

This section details the results of various convergence tests. Convergence results are run using the **twilight-zone** option, also known less formally as the **method of analytic solutions**. In this case the equations are forced so the the solution will be a known analytic function.

The tables show the maximum errors in the solution components. The rate shown is estimated convergence rate, σ, assuming $\mathrm{error} \propto h^{\sigma}$. The rate is estimated by a least squares fit to the data.

8.1 Incompressible Navier-Stokes convergence results

The 2D trigonometric solution used as a twilight zone function is

$$\begin{aligned}
u &= \frac{1}{2}\cos(\pi\omega_0 x)\cos(\pi\omega_1 y)\cos(\omega_3\pi t) + \frac{1}{2} \\
v &= \frac{1}{2}\sin(\pi\omega_0 x)\sin(\pi\omega_1 y)\cos(\omega_3\pi t) + \frac{1}{2} \\
p &= \cos(\pi\omega_0 x)\cos(\pi\omega_1 y)\cos(\omega_3\pi t) + \frac{1}{2}
\end{aligned}$$

The 3D trigonometric solution is

$$\begin{aligned}
u &= \cos(\pi\omega_0 x)\cos(\pi\omega_1 y)\cos(\pi\omega_2 z)\cos(\omega_3\pi t) \\
v &= \frac{1}{2}\sin(\pi\omega_0 x)\sin(\pi\omega_1 y)\cos(\pi\omega_2 z)\cos(\omega_3\pi t) \\
w &= \frac{1}{2}\sin(\pi\omega_0 x)\sin(\pi\omega_1 y)\sin(\pi\omega_2 z)\cos(\omega_3\pi t) \\
p &= \frac{1}{2}\sin(\pi\omega_0 x)\cos(\pi\omega_1 y)\cos(\pi\omega_2 z)\sin(\omega_3\pi t)
\end{aligned}$$

With have $\omega_0 == \omega_1 == \omega_2$ it follows that $\nabla \cdot \mathbf{u} = 0$. There are also algebraic polynomial solutions of different orders.

Tables (**??-??**) show results from running **OverBlown** on various grids.

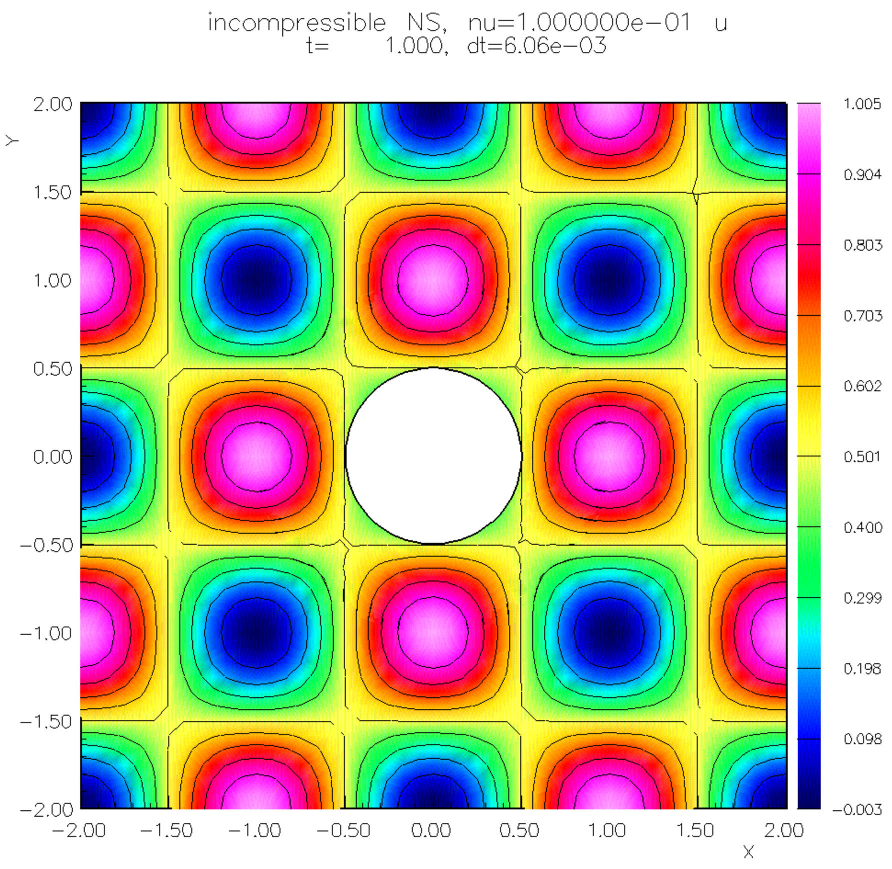

Figure 2: Incompressible N-S, twilight zone solution for convergence test

8.2 Compressible Navier-Stokes convergence results

grid	N	ρ	u	v	w	T
box5	5	6.3×10^{-2}	1.1×10^{-2}	2.0×10^{-2}	3.0×10^{-2}	2.6×10^{-3}
box10	10	2.1×10^{-2}	2.9×10^{-3}	5.2×10^{-3}	7.2×10^{-3}	7.0×10^{-4}
box20	20	5.7×10^{-3}	7.7×10^{-4}	1.3×10^{-3}	1.8×10^{-3}	1.8×10^{-4}
rate		1.7	1.9	2.0	2.0	1.9

Table 1: Compressible Navier-Stokes, $t = 1.0$, box, trigonometric TZ

grid	N	ρ	u	v	T
cic1	12	2.2×10^{-1}	2.4×10^{-1}	1.9×10^{-1}	6.0×10^{-2}
cic2	24	5.1×10^{-2}	6.4×10^{-2}	2.6×10^{-2}	1.0×10^{-2}
cic3	48	1.2×10^{-2}	1.3×10^{-2}	6.2×10^{-3}	2.5×10^{-3}
rate		2.1	2.1	2.5	2.3

Table 2: Compressible Navier-Stokes, $t = 1.0$, cic, trigonometric TZ

grid	N	ρ	u	v	T
sic1	1	1.4×10^{-1}	1.3×10^{-1}	8.2×10^{-2}	1.6×10^{-2}
sic2	2	4.4×10^{-2}	3.1×10^{-2}	2.0×10^{-2}	4.3×10^{-3}
sic3	4	1.0×10^{-2}	7.9×10^{-3}	4.9×10^{-3}	1.1×10^{-3}
rate		1.9	2.0	2.0	1.9

Table 3: Compressible Navier-Stokes, $t = 1.0$, sic, trigonometric TZ

grid	N	ρ	u	v	T
square5	5	4.5×10^{-2}	1.6×10^{-2}	3.0×10^{-2}	5.6×10^{-3}
square10	10	1.0×10^{-2}	4.1×10^{-3}	7.8×10^{-3}	1.4×10^{-3}
square20	20	2.5×10^{-3}	1.0×10^{-3}	1.9×10^{-3}	3.7×10^{-4}
square40	40	6.2×10^{-4}	2.6×10^{-4}	4.8×10^{-4}	9.2×10^{-5}
rate		2.1	2.0	2.0	2.0

Table 4: Compressible Navier-Stokes, $t = 1.0$, square, trigonometric TZ

9 A collection of some interesting examples

Here is a collection of interesting examples computed with the compressible **OverBlown** solver.

9.1 Mach 2 shock past a cylinder

Figure (3) shows a Mach 2 planar shock hitting a cylinder. The inviscid Euler equations are being solved using method CNSCAD.

9.2 Mach 2 shock past a triangle

Figure (4) shows a Mach 2 planar shock hitting a triangle. The triangle has slightly rounded corners and was defined by the `SmoothedPolygon` Mapping. The inviscid Euler equations are being solved using method CNSCAD. In this case we had to increase the coefficient of the second-order artificial viscosity to `av2=.5` to avoid negative pressures when the shock passes the corners at the trailing edge. This is probably due to the fact that the Mach number is very high near these corners after the shock has passed (the density is low here).

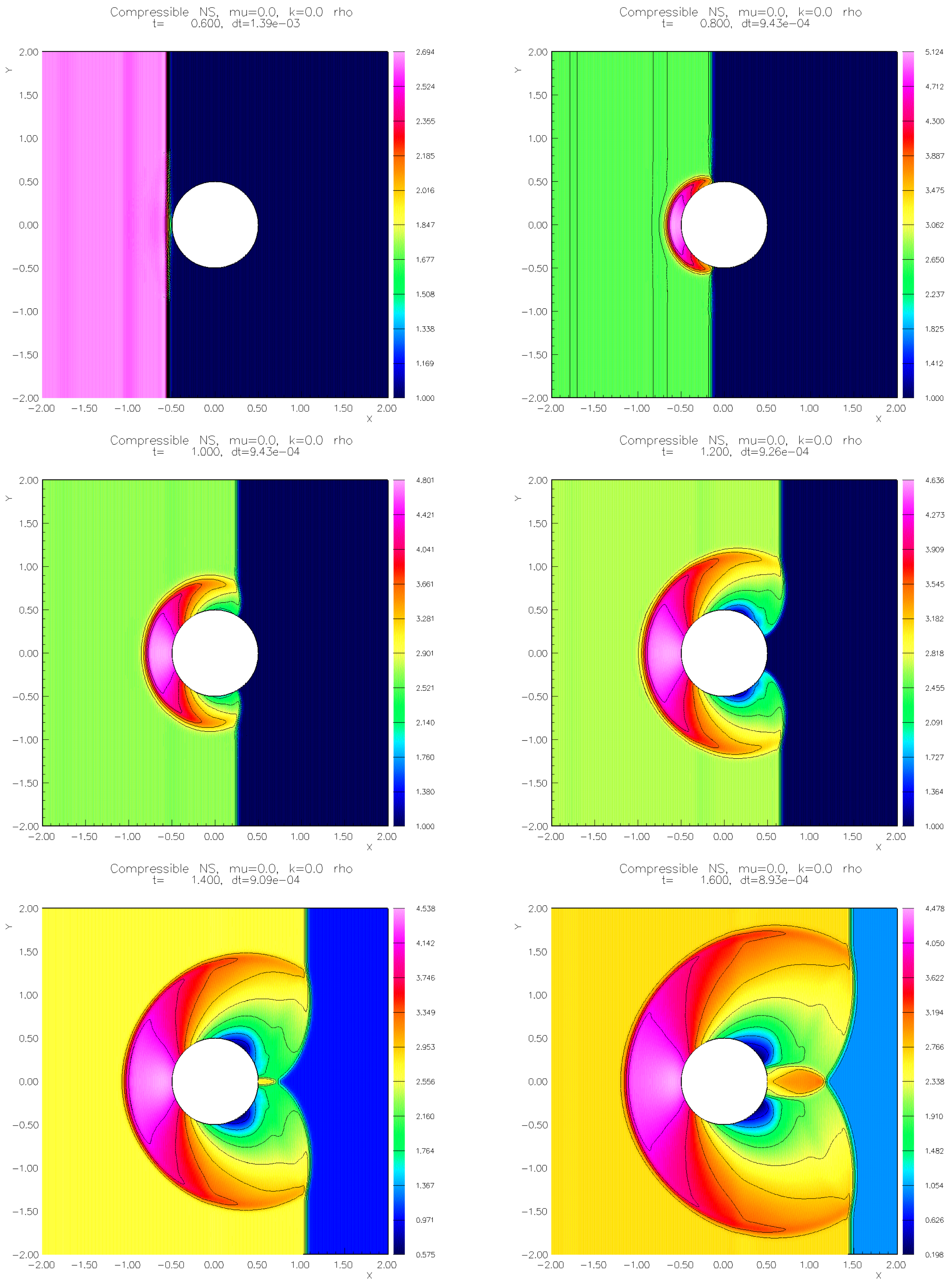

Figure 3: Mach 2 planar shock impinging on a cylinder.

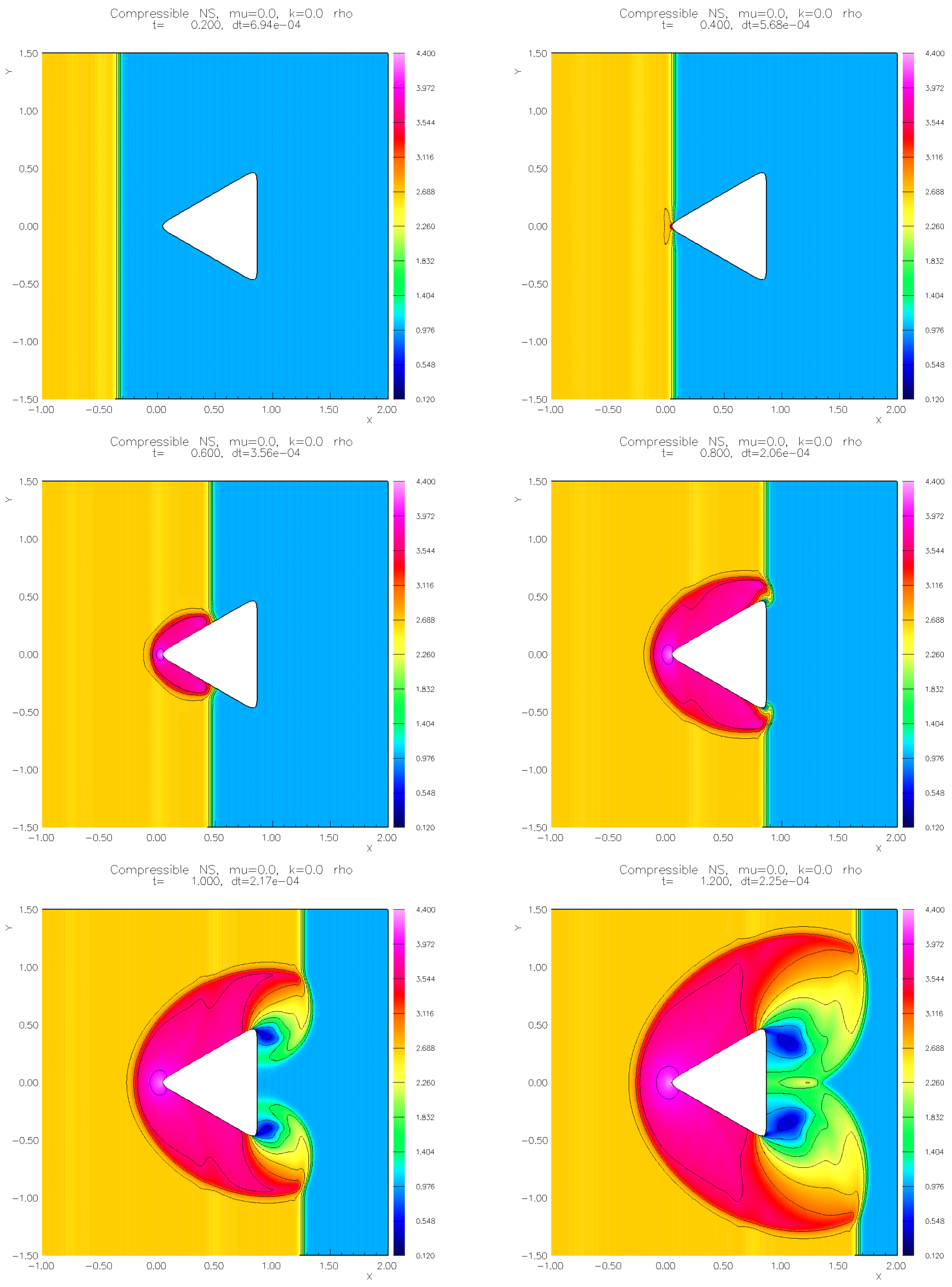

Figure 4: Mach 2 planar shock impinging on a triangle.

9.3 Mach 2 shock past a sphere

Figure (5) shows a Mach 2 planar shock hitting a sphere.

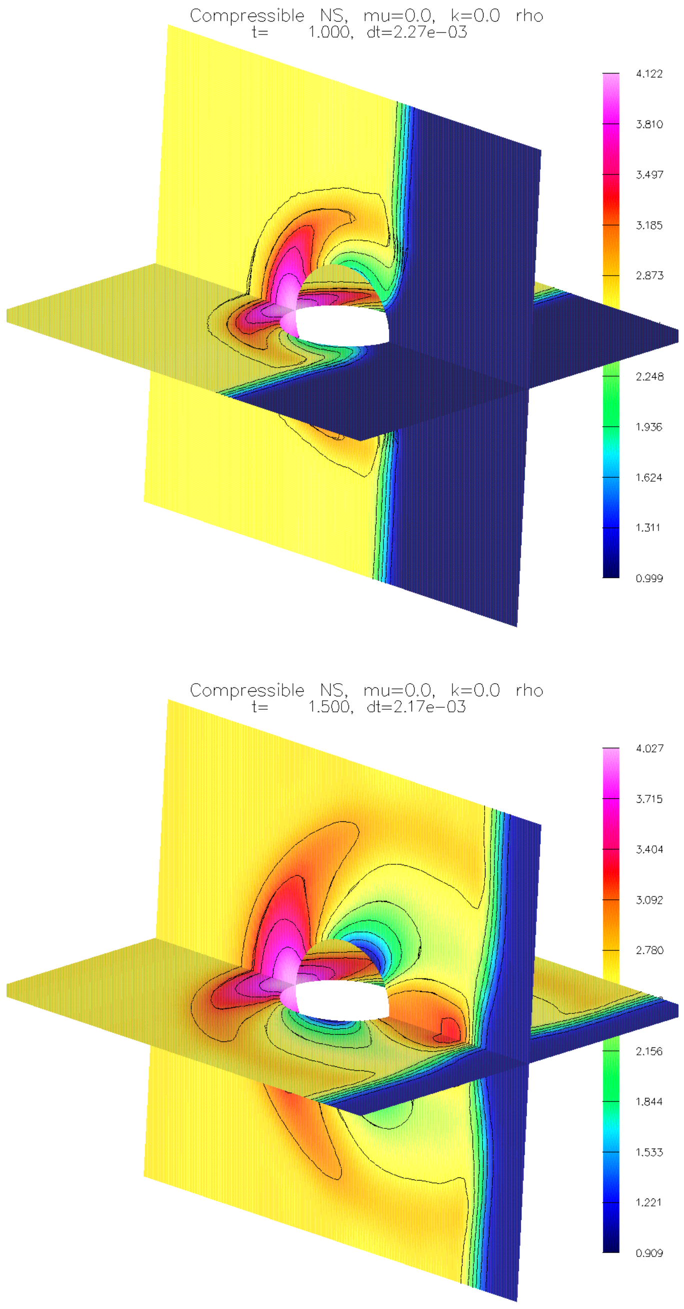

Figure 5: Mach 2 planar shock traveling past a sphere.

9.4 Shock travelling up a ramp

Figures (7), (8) shows a Mach 2 planar shock hitting a ramp.

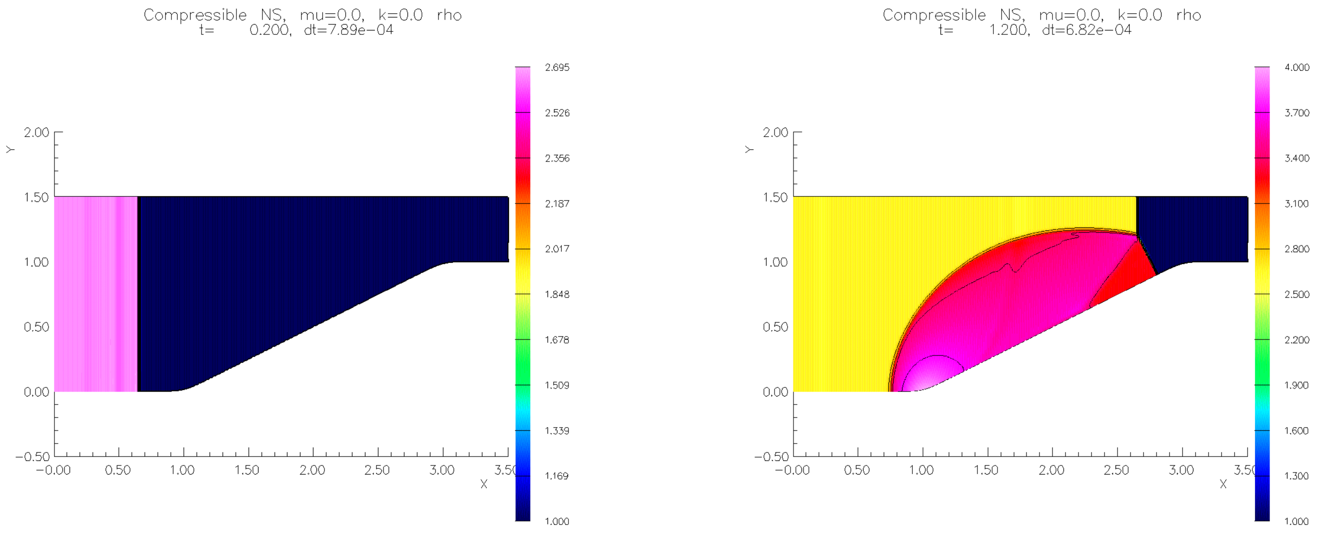

Figure 6: Mach 2 planar shock traveling up a ramp, density.

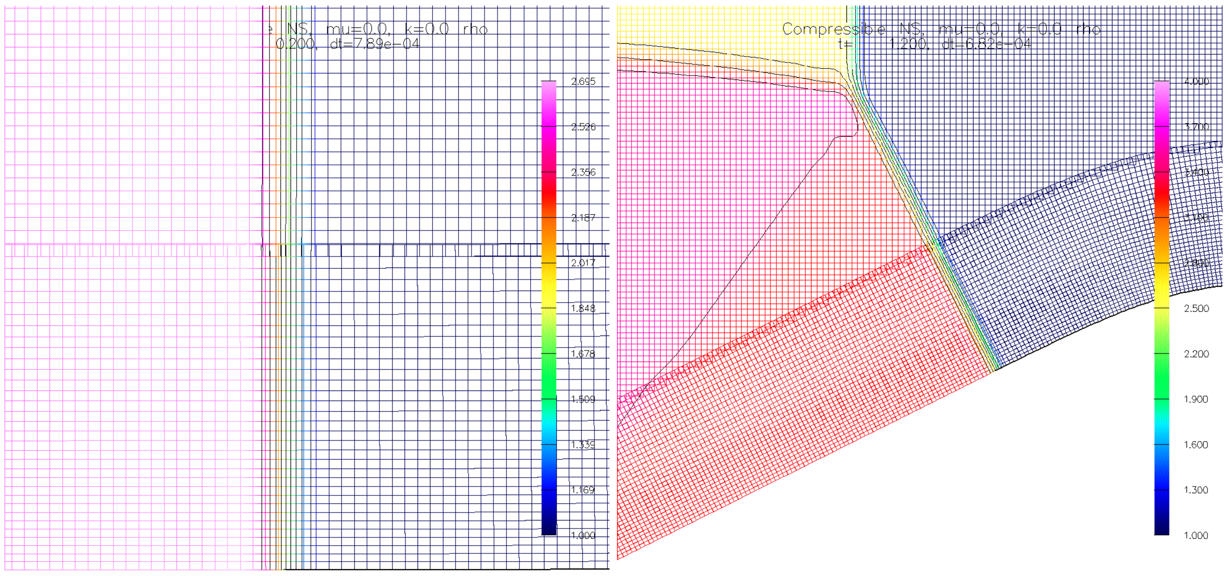

Figure 7: Mach 2 planar shock traveling up a ramp, magnified view of the region where the shock crosses the overlapping boundary.

References

[1] D. L. Brown, G. S. Chesshire, W. D. Henshaw, and D. J. Quinlan, *Overture: An object oriented software system for solving partial differential equations in serial and parallel environments*, in Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, 1997.

[2] D. L. Brown, W. D. Henshaw, and D. J. Quinlan, *Overture: An object oriented framework for solving partial differential equations*, in Scientific Computing in Object-Oriented Parallel Environments, Springer Lecture Notes in Computer Science, **1343**, 1997, pp. 177–194.

[3] G. Chesshire and W. Henshaw, *Conservation on composite overlapping grids*, IBM Research Report RC 16531, IBM Research Division, Yorktown Heights, NY, 1991.

[4] W. Henshaw, *Overture: An object-oriented system for solving PDEs in moving geometries on overlapping grids*, in First AFOSR Conference on Dynamic Motion CFD, June 1996, L. Sakell and D. Knight, eds., 1996, pp. 281–290.

[5] ——, *Mappings for Overture, a description of the Mapping class and documentation for many useful Mappings*, Research Report UCRL-MA-132239, Lawrence Livermore National Laboratory, 1998.

[6] ——, *Ogen: An overlapping grid generator for Overture*, Research Report UCRL-MA-132237, Lawrence Livermore National Laboratory, 1998.

[7] ——, *Oges user guide, a solver for steady state boundary value problems on overlapping grids*, Research Report UCRL-MA-132234, Lawrence Livermore National Laboratory, 1998.

[8] ——, *Plotstuff: A class for plotting stuff from Overture*, Research Report UCRL-MA-132238, Lawrence Livermore National Laboratory, 1998.

[9] ——, *OverBlown: A fluid flow solver for overlapping grids, user guide*, Research Report UCRL-MA-134288, Lawrence Livermore National Laboratory, 1999.

[10] ——, *OverBlownINS: The incompressible navier–stokes solver in OverBlown*, Research Report UCRL-MA-134289, Lawrence Livermore National Laboratory, 1999.

[11] A. JAMESON, *Numerical Methods in Fluid Dynamics*, vol. 1127 of Lecture Notes in Mathematics, Springer-Verlag, 1983, ch. Transonic Flow Calculations for Aircraft, pp. 156–242.

Index