Cgins Reference Manual: An Overture Solver for the Incompressible Navier–Stokes Equations on Composite Overlapping Grids,

William D. Henshaw[1]
Centre for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA, 94551.

July 10, 2014

LLNL-SM-455871

Abstract:

This is the reference guide for **Cgins**. Cgins is a program that can be used to solve incompressible fluid flow problems in complex geometry in two and three dimensions using composite overlapping grids. It is built upon the **Overture** object-oriented framework. Cgins solves the incompressible Navier-Stokes equations using a pressure-Poisson formulation. Second-order accurate and fourth-order accurate approximations are available. This reference guide describes in some detail the equations being solved, the discrete approximations, time-stepping methods, boundary conditions and convergence results. The reference guide also contains various notes related to different aspects of the equations. The reference guide concludes by providing a collection of interesting computations that have been performed with Cgins.

[1]This work was performed under the auspices of the U.S. Department of Energy (DOE) by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and by DOE contracts from the ASCR Applied Math Program.

Contents

1 Introduction

This is the reference guide for **Cgins**. Cgins is a program that can be used to solve incompressible fluid flow problems in complex geometry in two and three dimensions using composite overlapping grids. It is built upon the **Overture** object-oriented framework [1],[5],[2]. Cgins solves the incompressible Navier-Stokes equations using a pressure-Poisson formulation. Second-order accurate and fourth-order accurate approximations are available. Cgins can be used to

- solve problems in two and three dimensional complex domains,
- solve problems on moving grids (specified motion and rigid body motion),
- solve temperature dependent flows with buoyancy using the Boussinesq approximation,
- solve axisymmetric flows.

This reference guide describes in some detail the equations being solved, the discrete approximations, time-stepping methods, boundary conditions and convergence results. The reference guide also contains various notes related to different aspects of the equations. The reference guide concludes by providing a collection of interesting computations that have been performed with Cgins.
Other documents of interest that are available through the **Overture** home page are

- The Cgins User Guide [11] for getting started with Cgins and for descriptions of the various run-time options and parameters.
- The overlapping grid generator, `Ogen`, [7]. Use this program to make grids for cgins.
- 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 cgins to solve implicit time stepping equations and the Poisson equation for the pressure, [8].

2 The Equations

The incompressible Navier-Stokes equations are

$$\mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \nu\Delta\mathbf{u} + \mathbf{f}, \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0. \tag{2}$$

We solve the incompressible Navier-Stokes equations written in the form (pressure-poisson system)

$$\left.\begin{aligned} \mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \nu\Delta\mathbf{u} - \mathbf{f} &= 0 \\ \Delta p + (\nabla u\cdot\mathbf{u}_x + \nabla v\cdot\mathbf{u}_y + \nabla w\cdot\mathbf{u}_z) + C_d(\nu)\nabla\cdot\mathbf{u} - \nabla\cdot\mathbf{f} &= 0 \end{aligned}\right\} \quad \mathbf{x}\in\Omega$$
$$\left.\begin{aligned} B(\mathbf{u},p) &= 0 \\ \nabla\cdot\mathbf{u} &= 0 \end{aligned}\right\} \quad \mathbf{x}\in\partial\Omega \tag{3}$$
$$\mathbf{u}(\mathbf{x},0) = \mathbf{u}_0(\mathbf{x}) \qquad \text{at } t=0$$

There are n_d boundary conditions, $B(\mathbf{u},p)=0$, where n_d is the number of space dimensions. On a no-slip wall, for example, $\mathbf{u}=0$. In addition, a boundary condition is required for the pressure. The boundary condition $\nabla\cdot\mathbf{u}=0$ is added. With this extra boundary condition it follows that the above problem is equivalent to the formulation with the Poisson equation for the pressure replaced by $\nabla\cdot\mathbf{u}=0$ everywhere. The term $C_d(\nu)\nabla\cdot\mathbf{u}$ appearing in the equation for the pressure is used to damp the divergence [12]. For further details see also [4]

2.1 Addition of Temperature and Buoyancy: The Boussinesq Approximation

The effects of temperature and buoyancy can be modeled with the Boussinesq approximation given by

$$\begin{aligned}
\mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \nu\Delta\mathbf{u} + \alpha\mathbf{g}T - \mathbf{f} &= 0,\\
\Delta p - \mathcal{J}(\nabla\mathbf{u}) - \alpha(\mathbf{g}\cdot\nabla)T - \nabla\cdot\mathbf{f} &= 0,\\
T_t + (\mathbf{u}\cdot\nabla)T - k\Delta T - f_T &= 0,\\
\mathcal{J}(\nabla\mathbf{u}) \equiv (\nabla u\cdot\mathbf{u}_x + \nabla v\cdot\mathbf{u}_y + \nabla w\cdot\mathbf{u}_z)&\ .
\end{aligned}$$

Here T represents a temperature perturbation, $\mathbf{g}$ is the acceleration due to gravity, α is the coefficient of thermal expansivity and k is a coefficient of thermal conductivity, $k = \lambda/(\rho C_p)$.

The Rayleigh number and Prandtl number are defined as

$$Ra = \alpha g \Delta T d^3/(\nu\kappa), \qquad Pr = \nu/\kappa$$

where ΔT is the representative temperature difference and d is a length scale.

2.2 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,

$$U_t + UU_x + VU_r + p_x = \nu(U_{xx} + \frac{1}{r}(rU_r)_r \tag{4}$$

$$V_t + UV_x + VV_r + p_r = \nu(V_{xx} + \frac{1}{r}(rV_r)_r - \frac{V}{r^2}) \tag{5}$$

$$U_x + \frac{1}{r}(rV)_r = 0 \tag{6}$$

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 (Note that although the normal component of the velocity usually has even symmetry, $V=\mathbf{U}\cdot\hat{r}$ will have even symmetry at $r=0$ since $\hat{r}$ flips sign as we cross the axis)

$$\begin{aligned}U_r(x,0)&=0\\ V(x,0)&=0\quad,\quad V_r(x,0)=0\\ p_r(x,0)&=0\end{aligned}$$

In general all odd derivatives of $\mathbf{U}$ and 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) although the dirichlet condition $V(x,0)=0$ obviates the need for the former 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.

2.2.1 The pressure boundary condition for the axisymmetric case

The pressure boundary condition is formed from the normal component of the momentum equations. For a no-slip wall this becomes

$$p_n = \nu n_x\Big(U_{xx}+\frac{1}{r}(rU_r)_r\Big)+\nu n_r\Big(V_{xx}+V_{rr}+\frac{1}{r}V_r-\frac{V}{r^2}\Big)$$

We can eliminate some of the normal derivatives in this last expression (forming the equivalent of the *curl-curl* boundary conditions described in []) by taking derivative of the expression (6) for the divergence,

$$U_x+V_r+\frac{1}{r}V=0$$

giving

$$\begin{aligned}U_{xx} &= -V_{xr}-\frac{1}{r}V_x=0\\ V_{rr}+\frac{1}{r}V_r-\frac{1}{r^2}V &= -U_{xr}\end{aligned}$$

This leads to the pressure boundary condition

$$p_n = \nu n_x\Big(-V_{xr} + U_{rr} - \frac{1}{r}V_x + \frac{1}{r}U_r\Big) + \nu n_r\Big(V_{xx} - U_{xr}\Big) \tag{7}$$

On the axis, $r = 0$, this becomes

$$p_n = 2\nu n_x\Big(-V_{xr} + U_{rr}\Big) + \nu n_r\Big(V_{xx} - U_{xr}\Big), \qquad \text{(for } r = 0\text{)}.$$

3 Discretization

Let $\mathbf{V}_i$ and P_i denote the discrete approximations to $\mathbf{u}$ and p so that

$$\mathbf{V}_i \approx \mathbf{u}(\mathbf{x}_i) \quad , \quad P_i \approx p(\mathbf{x}_i)\ .$$

Here $\mathbf{V}_i = (V_{1i}, V_{2i}, V_{3i})$ and $i = (i_1, i_2, i_3)$ is a multi-index. After discretizing in space the equations we solve are of the form

$$\left.\begin{array}{rcl} \frac{d}{dt}\mathbf{V}_i + (\mathbf{V}_i\cdot\nabla_h)\mathbf{V}_i + \nabla_h P_i - \nu\Delta_h\mathbf{V}_i - \mathbf{f}(\mathbf{x}_i,t) &=& 0 \\ \Delta_h P_i - \sum_m \nabla_h V_{m,i}\cdot D_{m,h}\mathbf{V}_i - C_{d,i}\nabla_h\cdot\mathbf{V}_i - \nabla_h\cdot\mathbf{f}(\mathbf{x}_i,t) &=& 0 \end{array}\right\} \quad \mathbf{x}\in\Omega$$

$$\left.\begin{array}{rcl} B(\mathbf{V}_i, P_i) &=& 0 \\ \nabla_h\cdot\mathbf{V}_i &=& 0 \end{array}\right\} \quad \mathbf{x}_i \in \partial\Omega_h$$

$$\mathbf{V}(\mathbf{x}_i, 0) = \mathbf{U}_0(\mathbf{x}_i) \qquad \text{at } t = 0$$

where the divergence damping coefficient, $C_{d,i}$ is defined below. The subscript "h" denotes a second or fourth-order centred difference approximation,

$$D_{m,h} \approx \frac{\partial}{\partial x_m} \quad , \quad \nabla_h = (D_{1,h}, D_{2,h}, D_{3,h}) \quad , \quad \Delta_h \approx \sum_m \frac{\partial^2}{\partial x_m^2}$$

Extra numerical boundary conditions are also added, see [4] [15] for further details. An artificial diffusion term can be added to the momentum equations. This is described in section (6.1).

When discretized in space on an overlapping grid this system of PDEs can be thought of as a large system of ODEs of the form

$$\frac{d\mathbf{U}}{dt} = \mathcal{F}(t, \mathbf{U}, \mathbf{P})$$

where $\mathbf{U}$ is a vector of all solution values at all grid points. For the purpose of discussing time-stepping methods it is often convenient to think of the pressure as simply a function of $\mathbf{U}$, $\mathbf{P} = \mathcal{P}(\mathbf{U})$ There are also interpolation equations that need to be satisfied but this causes no difficulties.

4 Dimensional Units and Non-dimensional Equations

When you solve the incompressible Navier-Stokes equations with Cgins it solves the following equations

$$\mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \nu\Delta\mathbf{u}, \tag{8}$$

$$\nabla\cdot\mathbf{u} = 0. \tag{9}$$

There is only one parameter that you specify and that is ν, the kinematic viscosity. Cgins does NOT scale the input values that you specify for the initial conditions and boundary conditions. Cgins does not scale the dimensions of the grid. Also note that the pressure p is only determined up to a constant.

Let's relate these equations to the equations in dimensional form,

$$\mathbf{u}^d_{t^d} + (\mathbf{u}^d\cdot\nabla^d)\mathbf{u}^d + \frac{1}{\rho^d}\nabla^d p^d = \frac{\mu}{\rho^d}\Delta^d\mathbf{u}^d, \tag{10}$$

$$\nabla^d\cdot\mathbf{u}^d = 0. \tag{11}$$

Here the density ρ^d is a constant and μ is the constant viscosity. The superscript d denotes dimensional units. For example, the components of $\mathbf{u}^d$ might be measured in m/s, lengths in m, the pressure p^d in N/m^2 and the viscosity coefficient μ in $kg/(m\ s)$.

Suppose that you want to use dimensional units, for example MKS units, to specify a problem for Cgins. The grid should then be built with dimensions in m. The inflow values for the velocity should be given in m/s. The kinematic viscosity should be set equal to $\nu = \mu/\rho^d$ which has units m^2/s. Comparing equation (10) to equation (8) we see that the pressure computed by Cgins will be $p = p^d/\rho^d$. Thus you should specify boundary conditions and initial conditions for p from $p = p^d/\rho^d$ (which has units of m^2/s^2).

In general one may want to non-dimensionalize the equations before solving them with Cgins. Let us choose appropriate values, U, L, R, to scale the length, velocity and density, and define non-dimensional variables

$$\mathbf{x}^n = \mathbf{x}^d/L\ , \qquad \rho^n = \rho^d/R\ , \tag{12}$$

$$t^n = t^d/(L/U)\ , \qquad \mathbf{u}^n = \mathbf{u}^d/U\ , \tag{13}$$

$$p^n = p^d/(RU^2)\ . \tag{14}$$

The Navier-Stokes equations then become

$$\mathbf{u}^n_{t^n} + (\mathbf{u}^n\cdot\nabla^n)\mathbf{u}^n + \frac{1}{\rho^n}\nabla^n p^n = \frac{\mu}{\rho^n RUL}\Delta^n\mathbf{u}^n, \tag{15}$$

$$\nabla^n\cdot\mathbf{u}^n = 0. \tag{16}$$

In this case we will generate the grid with lengths that have been scaled by L. We will specify inflow velocities with values scaled by U, the pressure will be scaled by RU^2, the kinematic viscosity will be defined by $\nu = \mu/(\rho^n RUL)$ and the non-dimensional pressure p^n will be related to the computed value p by $p = p^n/\rho^n$.

5 Time stepping methods

5.1 Adams Predictor-Corrector

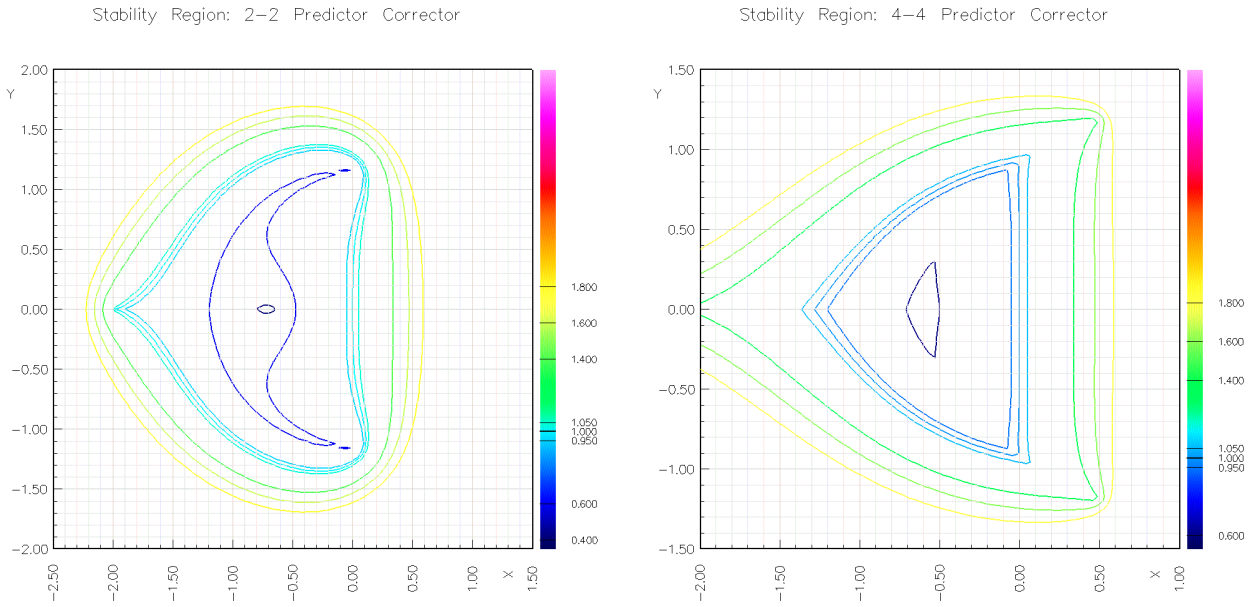

Figure 1: Stability regions for the predictor corrector methods. Left: second-order method, PECE mode. Right: fourth-order method, PECE mode.

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

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

(see for example Lambert[16]). The stability region for this method is shown in figure (1).

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

$$\frac{\mathbf{u}^p - \mathbf{u}^n}{\Delta t} = p_0\mathbf{f}^n + p_1\mathbf{f}^{n-1} + p_2\mathbf{f}_{n-2} + p_3\mathbf{f}_{n-3}$$
$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = c_0\mathbf{f}^p + c_1\mathbf{f}^n + c_2\mathbf{f}^{n-1} + c_3\mathbf{f}_{n-2}$$
$$\begin{aligned}
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 $\Delta t_m = t_{n+1-m} - t_{n-m}$, $m = 0, 1, 2, 3$.

5.2 Implicit multistep method with viscous terms implicit

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.

5.3 Implicit multistep method with the viscous and non-linear terms implicit

Consider now the case of treating the viscous and non-linear terms in an implicit manner.

In the general case we are solving a nonlinear equation of the form

$$\mathbf{u}_t = \mathbf{f}(\mathbf{u}).$$

We linearize a part of the operator $\mathbf{f}(\mathbf{u})$ around the state $\mathbf{u}^*(\mathbf{x},t)$ and write the equation as

$$\begin{aligned}\mathbf{u}_t &= L(\mathbf{u};\mathbf{u}^*) + \big(\mathbf{f}(\mathbf{u}) - L(\mathbf{u};\mathbf{u}^*)\big) &(17)\\ &= L(\mathbf{u};\mathbf{u}^*) + \mathbf{f}_E(\mathbf{u};\mathbf{u}^*) &(18)\end{aligned}$$

where

$$\mathbf{f}_E(\mathbf{u};\mathbf{u}^*) \equiv \mathbf{f}(\mathbf{u}) - L(\mathbf{u};\mathbf{u}^*) \tag{19}$$

Here $L(\mathbf{u};\mathbf{u}^*)$ is a linear operator of $\mathbf{u}$ that depends upon $\mathbf{u}^*$. For example if $\mathbf{f}(\mathbf{u}) = (\mathbf{u}\cdot\nabla)\mathbf{u}$ then one choice could be $L(\mathbf{u};\mathbf{u}^*) = (\mathbf{u}^*\cdot\nabla)\mathbf{u}$. Another choice could be $L(\mathbf{u};\mathbf{u}^*) = (\mathbf{u}^*\cdot\nabla)\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}^*$. We can now define a implicit time-stepping method that uses the form (18). A backward-Euler approximation to the full equations is

$$\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t} = L(\mathbf{u}^{n+1};\mathbf{u}^*) + \mathbf{f}_E(\mathbf{u}^{n+1};\mathbf{u}^*)\ .$$

This equation can be solved by *implicit* iteration (treating the L term implicitly),

$$\frac{\mathbf{v}^{k}-\mathbf{u}^n}{\Delta t} = L(\mathbf{v}^{k};\mathbf{u}^*) + \mathbf{f}_E(\mathbf{v}^{k-1};\mathbf{u}^*)$$

where we take $\mathbf{v}^0 = \mathbf{u}^n$ and iterate on $k = 0, 1, 2, ...$ for some number of steps. If we iterate until convergence then we will have solved the original equations with backward-Euler, independent of the choice L and $\mathbf{u}^*$,

$$\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t} = \mathbf{f}(\mathbf{u}^{n+1})\ .$$

Of course for best convergence for a large value of Δt, we want make good choices for L and $\mathbf{u}^*$.

We could also solve the equations with the trapezodial like rule

$$\begin{aligned}\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t} &= \alpha\mathbf{f}(\mathbf{u}^{n+1}) + (1-\alpha)\mathbf{f}(\mathbf{u}^n)\\ &= \alpha L(\mathbf{u}^{n+1};\mathbf{u}^*) + (1-\alpha)L(\mathbf{u}^n;\mathbf{u}^*) + \alpha\mathbf{f}_E(\mathbf{u}^{n+1};\mathbf{u}^*) + (1-\alpha)\mathbf{f}_E(\mathbf{u}^n;\mathbf{u}^*),\\ &= \alpha L(\mathbf{u}^{n+1};\mathbf{u}^*) + \alpha\mathbf{f}_E(\mathbf{u}^{n+1};\mathbf{u}^*) + (1-\alpha)\mathbf{f}(\mathbf{u}^n)\\ &= \alpha L(\mathbf{u}^{n+1};\mathbf{u}^*) + \alpha\big(\mathbf{f}(\mathbf{u}^{n+1}) - L(\mathbf{u}^{n+1};\mathbf{u}^*)\big) + (1-\alpha)\mathbf{f}(\mathbf{u}^n)\end{aligned}$$

where $\alpha = \frac{1}{2}$ is the trapezodial rule and $\alpha = 1$ is backward-Euler. This equation can be solved by the implicit iteration

$$\frac{\mathbf{v}^{k}-\mathbf{u}^n}{\Delta t} = \alpha L(\mathbf{v}^{k};\mathbf{u}^*) + \alpha\big(\mathbf{f}(\mathbf{v}^{k-1}) - L(\mathbf{v}^{k-1};\mathbf{u}^*)\big) + (1-\alpha)\mathbf{f}(\mathbf{u}^n)$$

For $\alpha = 1/2$, the first iterate $\mathbf{v}^1$ is second-order accurate in the implicit part L and first order accurate in the explicit part $\mathbf{f}_E$. If $\mathbf{f}_E(\mathbf{u}^n;\mathbf{u}^*) = \mathcal{O}(\Delta t)$ then the $\mathbf{v}^1$ is second-order accurate. For $\alpha = 1/2$ the second and later iterates, $\mathbf{v}^k$, $k \ge 2$, should be second-order accurate.

To implement this scheme we need to (1) build the matrix M for the implicit operator,

$$M\mathbf{v}^k = (I - \alpha\Delta t A(\mathbf{u}^*))\mathbf{v}^k = \mathbf{v}^k - \alpha\Delta t L(\mathbf{v}^k;\mathbf{u}^*)$$

(2) evaluate the right-hand-side $\mathbf{f}(\mathbf{v})$, and (3) evaluate the linearized operator $L(\mathbf{v};\mathbf{u}^*)$ or the combination $\mathbf{f}(\mathbf{v}) - L(\mathbf{v};\mathbf{u}^*)$.

We could use a predictor step with an second-order accurate explicit predictor for $\mathbf{f}_E$,

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

With $\alpha = 1/2$ the predicted value $\mathbf{u}^p$ would be second order.

5.4 Example linearizations

Here is an example for the INS:

$$\begin{aligned}
\mathbf{f}(\mathbf{u}) &= \nu\Delta\mathbf{u} - (\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla p \\
L(\mathbf{u};\mathbf{u}^*) &= \nu\Delta\mathbf{u} - \Big((\mathbf{u}^*\cdot\nabla)\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}^*\Big) \\
\mathbf{f}_E(\mathbf{u};\mathbf{u}^*) &= \mathbf{f}(\mathbf{u}) - L(\mathbf{u};\mathbf{u}^*) \\
&= -(\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla p + \Big((\mathbf{u}^*\cdot\nabla)\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}^*\Big) \\
&= -\nabla p - ((\mathbf{u}-\mathbf{u}^*)\cdot\nabla)(\mathbf{u}-\mathbf{u}^*) + (\mathbf{u}^*\cdot\nabla)\mathbf{u}^*
\end{aligned}$$

When the viscosity is a function of $\mathbf{u}$, $\nu = \nu(\mathbf{u})$, we could use Here is an example for the INS:

$$\begin{aligned}
\mathbf{f}(\mathbf{u}) &= \nabla\cdot(\nu(\mathbf{u})\nabla\mathbf{u}) - (\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla p \\
L(\mathbf{u};\mathbf{u}^*) &= \nabla\cdot(\nu(\mathbf{u}^*)\nabla\mathbf{u}) + \nabla\cdot(\nu(\mathbf{u})\nabla\mathbf{u}^*) - \Big((\mathbf{u}^*\cdot\nabla)\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}^*\Big) \\
\mathbf{f}_E(\mathbf{u};\mathbf{u}^*) &= \mathbf{f}(\mathbf{u}) - L(\mathbf{u};\mathbf{u}^*) \\
&= \nabla\cdot(\nu(\mathbf{u})\nabla\mathbf{u}) - \nabla\cdot(\nu(\mathbf{u}^*)\nabla\mathbf{u}) - \nabla\cdot(\nu(\mathbf{u})\nabla\mathbf{u}^*) \\
&\quad - (\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla p + \Big((\mathbf{u}^*\cdot\nabla)\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}^*\Big) \\
&= \nabla\cdot((\nu(\mathbf{u}) - \nu(\mathbf{u}^*))\nabla(\mathbf{u}-\mathbf{u}^*)) - \nabla\cdot(\nu(\mathbf{u}^*)\nabla\mathbf{u}^*) \\
&\quad - \nabla p - ((\mathbf{u}-\mathbf{u}^*)\cdot\nabla)(\mathbf{u}-\mathbf{u}^*) + (\mathbf{u}^*\cdot\nabla)\mathbf{u}^*
\end{aligned}$$

5.5 Variable time stepping

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

6 Divergence Damping

The divergence damping term, $C_{d,i}\nabla_h \cdot \mathbf{V}_i$, appears in the pressure equation. In simplified terms, the coefficient C_d is taken proportional to the inverse of the time step, $C_d \sim \frac{1}{\Delta t}$. In practice we have found better results by taking $C_d \sim \frac{\nu}{\Delta x^2}$. For explicit time stepping theses are very similar since the explicit time step restriction is something like $\frac{\nu \Delta t}{\Delta x^2} < C$. To allow for the case $\nu = 0$ we use the minumum grid spacing, $h_{\min}$, instead of ν, if $h_{\min} > \nu$. The size of C_d affects the time step, the stability condition is proportional to $C_d \Delta t$. As a result we do not want C_d to be much larger than $1/\Delta t$, and thus it is limited by $\frac{C_t}{\Delta t}$ where C_t is a constant with default value of 0.25. (Note that we don't actually know the true Δt at this point, it depends on C_d, so we just use a guess).

Here is the actual formula for the divergence damping coefficient:

$$C_{d,i} = \min(\mathcal{D}_i, \frac{C_t}{\Delta t})$$

where

$$\begin{aligned}
\mathcal{D}_i &= C_0 \ \max(\nu, h_{\min}) \ \left(\frac{1}{(\Delta_{0,r_1} x_{1,i})^2} + \frac{1}{(\Delta_{0,r_2} x_{2,i})^2} + \frac{1}{(\Delta_{0,r_3} x_{3,i})^2} \right) \\
\Delta_{0,r_1} x_{m,i} &= \frac{1}{2}(x_{m,i_1+1} - x_{m,i_1-1}) \qquad \text{(undivided second difference)} \\
h_{\min} &= \min_i(\|\Delta_{+,r_1}\mathbf{x}_i\|, \Delta_{+,r_2}\mathbf{x}_i\|, \Delta_{+,r_3}\mathbf{x}_i\|) \qquad \text{(minimum grid spacing)}
\end{aligned}$$

and where $C_0 = 1.$ by default.

6.1 Artificial Diffusion

Cgins implements artificial diffusions based either on a second-order undivided difference or a fourth-order undivided difference.

The artificial diffusions are

$$\mathbf{d}_{2,i} = (\texttt{ad21} + \texttt{ad22}|\nabla_h \mathbf{V}_i|_1) \sum_{m=1}^{n_d} \Delta_{m+}\Delta_{m-}\mathbf{V}_i \tag{20}$$

in the second-order case and

$$\mathbf{d}_{4,i} = -(\texttt{ad41} + \texttt{ad42}|\nabla_h \mathbf{V}_i|_1) \sum_{m=1}^{n_d} \Delta_{m+}^2\Delta_{m-}^2\mathbf{V}_i \tag{21}$$

in the fourth-order case. Here $|\nabla_h \mathbf{V}_i|_1$ is the magnitude of the gradient of the velocity and $\Delta_{m\pm}$ are the forward and backward undivided difference operators in direction m

$$\begin{aligned}
|\nabla_h \mathbf{V}_i|_1 &= n_d^{-2} \textstyle\sum_{m=1}^{n_d} \sum_{n=1}^{n_d} |D_{m,h} V_{ni}| \\
\Delta_{1+}\mathbf{V}_i &= \mathbf{V}_{i_1+1} - \mathbf{V}_i \\
\Delta_{1-}\mathbf{V}_i &= \mathbf{V}_i - \mathbf{V}_{i_1+1} \\
\Delta_{2+}\mathbf{V}_i &= \mathbf{V}_{i_2+1} - \mathbf{V}_i \\
\Delta_{2-}\mathbf{V}_i &= \mathbf{V}_i - \mathbf{V}_{i_2+1} \quad \text{etc.}
\end{aligned}$$

The artificial diffusion is added to the momentum equations

$$\frac{d}{dt}\mathbf{V}_i + (\mathbf{V}_i \cdot \nabla_h)\mathbf{V}_i + \nabla_h P_i - \nu\Delta_h \mathbf{V}_i - \mathbf{f}(\mathbf{x}_i, t) - \mathbf{d}_{m,i} = 0$$

but does not change the pressure equation. Typical choices for the constants $\texttt{ad21} = \texttt{ad41} = 1$ and $\texttt{ad22} = \texttt{ad42} = 1.$. These artificial diffusions should not affect the order of accuracy of the method. With the artificial diffusion turned on to a sufficient degree, the real viscosity can be set at low as zero, $\texttt{nu} = 0$.

In difficult cases you may need to increase the coefficients $\texttt{ad21}$ and $\texttt{ad22} = 1.$ to keep the solution stable. I suggest trying $\texttt{ad21} = \texttt{ad22} = 2.$ then $4.$ etc. until the solution doesn't blow up. I don't think I have ever needed a value larger than 10..

This form of the artificial diffusion is based on a theoretical result [13][14] that states that the minimum scale, $\lambda_{\min}$, of solutions to the incompressible Navier-Stokes equations is proportional to the square root of the kinematic viscosity divided by the square root of the maximum velocity gradient:

$$\lambda_{\min} \propto \sqrt{\frac{\nu}{|\nabla \mathbf{u}| + c}} \ .$$

This result is valid locally in space so that $|\nabla \mathbf{u}|$ measures the local value of the velocity gradient. The minimum scale measures the size of the smallest eddy or width of the sharpest shear layer as a function of the viscosity and the size of the gradients of $\mathbf{u}$. Scales smaller than the minimum scale are in the exponentially small part of the spectrum.

This result can be used to tell us the smallest value that we can choose for the (artificial) viscosity, ν_A, and still obtain a reasonable numerical solution. We require that the artificial viscosity be large enough so that the smallest (but still significant) features of the flow are resolved on the given mesh. If the local grid spacing is h, then we need

$$h \propto \sqrt{\frac{\nu_A}{|\nabla \mathbf{u}| + c}} \ .$$

This gives

$$\nu_A = (c_1 + c_2|\nabla \mathbf{u}|)h^2$$

and thus we can choose an artificial diffusion of

$$(c_1 + c_2|\nabla \mathbf{u}|)h^2 \Delta \mathbf{u}$$

which is just the form (20).

In the fourth-order accurate case we wish to add an artificial diffusion of the form

$$-\nu_A \Delta^2 \mathbf{u}$$

since, as we will see, this will lead to $\nu_A \propto h^4$. In this case, if we consider solutions to the incompressible Navier-Stokes equations with the diffusion term $\nu \Delta \mathbf{u}$ replaced by $-\nu_A \Delta^2 \mathbf{u}$ then the minimum scale would be

$$\lambda_{\min} \propto \left(\frac{\nu_A}{|\nabla \mathbf{u}|} \right)^{1/4}$$

Following the previous argument leads us to choose an artificial diffusion of the form

$$-(c_1 + c_2|\nabla \mathbf{u}|)h^4 \Delta^2 \mathbf{u}$$

which is just like (21).

7 Boundary Conditions

The boundary conditions for method INS are

$$\text{noSlipWall} = \begin{cases} \mathbf{u} = \mathbf{g} & \text{velocity specified} \\ \nabla \cdot \mathbf{u} = 0 & \text{divergence zero} \end{cases}$$

$$\text{slipWall} = \begin{cases} \mathbf{n} \cdot \mathbf{u} = g & \text{normal velocity specified} \\ \partial_n (\mathbf{t}_m \cdot \mathbf{u}) = 0 & \text{normal derivative of tangential velcity is zero} \\ \nabla \cdot \mathbf{u} = 0 & \text{divergence zero} \end{cases}$$

$$\text{inflowWithVelocityGiven} = \begin{cases} \mathbf{u} = \mathbf{g} & \text{velocity specified} \\ \partial_n p = 0 & \text{normal derivative of the pressure zero.} \end{cases}$$

$$\text{outflow} = \begin{cases} \text{extrapolate } \mathbf{u} & \\ \alpha p + \beta \partial_n p = g & \text{mixed derivative of p given.} \end{cases}$$

$$\text{symmetry} = \begin{cases} \mathbf{n} \cdot \mathbf{u} \text{: odd}, \mathbf{t}_m \cdot \mathbf{u} \text{: even} & \text{vector symmetry} \\ \partial_n p = 0 & \text{normal derivative of the pressure zero.} \end{cases}$$

$$\text{dirichletBoundaryCondition} = \begin{cases} \mathbf{u} = \mathbf{g} & \text{velocity specified} \\ p = P & \text{pressure given} \end{cases}$$

8 Projecting the initial conditions

In this section we discuss the ‘‘`project initial conditions`’’ option that is often used with Cgins in order to adjust the user specfied initial conditions.

The equation $\nabla \cdot \mathbf{u} = 0$ imposes some compatibility constraints on the initial and boundary conditions. For example, the initial conditions should satisfy $\nabla \cdot \mathbf{u}_0 = 0$. Imposing the divergence free condition up to the boundary implies that that the normal component of $\mathbf{u}_0$ should equal the normal component of the velocity specified on the boundary, $\mathbf{n} \cdot \mathbf{u}_0 = \mathbf{n} \cdot \mathbf{u}_{\partial\Omega}$, see [3]. A further constraint follows by integrating $\nabla \cdot \mathbf{u}$ over Ω and using the Gauss divergence theorem, giving $\int_{\partial\Omega} \mathbf{n} \cdot \mathbf{u} ds = 0$.

In many cases it may not be easy to generate initial conditions that satisfy the compatibility conditions. It is, however, well known how to take a given function $\mathbf{u}_I$ and to project this function so that it is divergence free. This projection is defined by

$$\begin{aligned} \mathbf{u}_0 &= \mathbf{u}_I + \nabla\phi \quad \mathbf{x} \in \Omega \\ \mathbf{u}_0 &= \mathbf{u}_I \quad \mathbf{x} \in \partial\Omega \\ \Delta\phi &= -\nabla \cdot \mathbf{u}_I \quad \mathbf{x} \in \Omega \\ \mathbf{n} \cdot \nabla\phi &= 0 \quad \mathbf{x} \in \partial\Omega \end{aligned}$$

The function $\mathbf{u}_0$ will satisfy $\nabla \cdot \mathbf{u}_0 = 0$ and $\mathbf{n} \cdot \mathbf{u}_0$ will equal $\mathbf{n} \cdot \mathbf{u}_I$ on the boundary. Note, however, that this projection does not force the tangential components of $\mathbf{u}_0$ to be continuous at the boundary.

In the discrete case this projection is easy to compute since the function ϕ satisfies the same equation as the pressure, but with different data. However, the discrete projection operator does not make the discrete divergence exactly zero for the same reason that the discrete divergence is not exactly zero in the overall scheme. Thus if the initial function, $\mathbf{u}_I$, is not smooth then it may be necessary to first smooth $\mathbf{u}_I$ before applying the projection. In practice a sequence of smoothing and projection steps is applied until the discrete divergence nolonger decreases significantly.

9 Specification of body forces and boundary forcing

In this section we describe how to define *body forces* and *boundary forcings*. *Body forces* are volume forcing terms that are added to the right-hand-side of the PDE's (momentum equations, temperature equation etc.). *Boundary forcings* are right-hand-sides added to the boundary conditions.
Examples of body forces are

1. add a heat source to a given region of the domain.
2. add a drag force to a given region of the domain.
3. define an immersed boundary over some region. This option provides an approximate way to add new geometry (e.g. small or complicated features) to a problem without having to build a proper overlapping grid for the features.
4. define a wake model such as an actuator disk (e.g. modeling a fan).
5. change material parameters such as thermal conductivity, heat capacity or viscosity over a given region (needs to be finished).

Examples of boundary forcings are:

1. specify the temperature or heat flux on a sub-region of a grid face.
2. add a local inflow region with parabolic profile to a portion of a grid face.
3. define a temperature boundary condition that varies from Dirichlet (isothermal) to Neumann (adiabatic) or mixed over a given grid face.

The construction of a body/boundary force requires the specification of

Region : specify a region (e.g. box, cylinder, triangulated surface) over which the body force applies.

Parameters : specify parameters that define the force (e.g. inflow velocity, temperature, heat flux)

Profile : define a spatial profile for the force (e.g. parabolic inflow).

Time variation : define the time variation of the force (e.g. define a time varying heat flux on a sub-region of a boundary).

In addition, a given body force (e.g. drag force or actuator disk) will have parameters associated with it (e.q. drag law).

Note: If multiple forcings are defined for the same volume or boundary, the corresponding forces are (by default) added together.

9.1 Defining body-force and boundary-force regions

Forcing regions over which body/boundary forces are applied can be defined using one or more of the following

Box : a box (square in two-dimensions).

Ellipse : an ellipse (or circle).

mask from grid function : the body is defined by a grid function *mask* that contains values indicating whether a grid point is inside or outside the body. Ideally this should be a signed distance function with negative values being inside the body.

Mapping : use this option to define an arbitrary curve in 2D or a surface in 3D (described in more detail below).

Note: when defining regions for boundary forcings, the actual boundary should be enclosed in the region. Thus in two-dimensions, to define a boundary forcing on an interval $[x_a, x_b]$ of a horizontal wall at $y = y_w$ one should define a narrow box (square) to enclose the interval, $B = [x_a, x_b] \times [y_w - \epsilon, y_w + \epsilon]$.

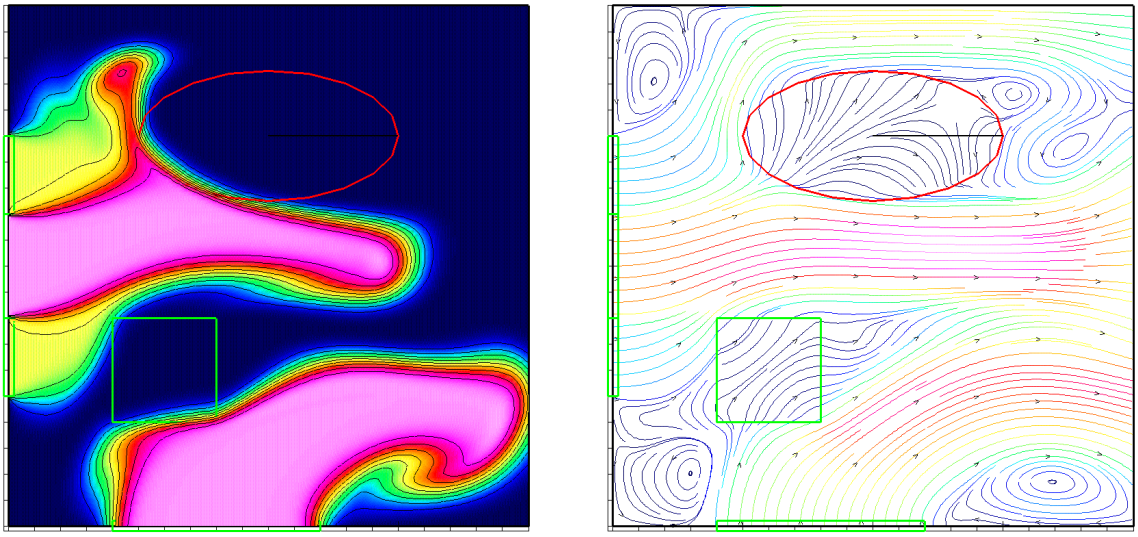

Figure 2: This figure, temperature on left and stream-lines on the right, illustrates uses of various body-forces and boundary-forcings. Boundary-forces are used to define inflow regions on the left and bottom walls. Immersed-boundary body forces are used to define a square obstacle and elliptical obstacle.

2D regions defined by a curve: Very general body force regions in 2D can be defined from Overture Mappings that can define closed curves in 2D (e.g. `AirfoilMapping`, `NurbsMapping`, `SplineMapping`). The determination of whether a point is inside and outside the region is determined with a fast ray-tracing algorithm. Figure 3 shows examples of 2D body force regions defined from Mappings.

3D regions defined by surface triangulations: Very general body force regions in 3D can be defined as an unstructured surface. The `UnstructuredMapping` class can be used to represent these surfaces. Input formats include the popular STL file format (STereoLithography or Standard Tessellation Language) and PLY file format (Polygon File Format or the Stanford Triangle Format). Figure 5 shows examples of some surfaces. The `UnstructuredMapping` has a fast algorithm for determining whether a given point in space is inside or outside of the surface (this requires the surface to be *water-tight*. The algorithm uses an alternating-digit-tree (ADT) and ray-tracing.

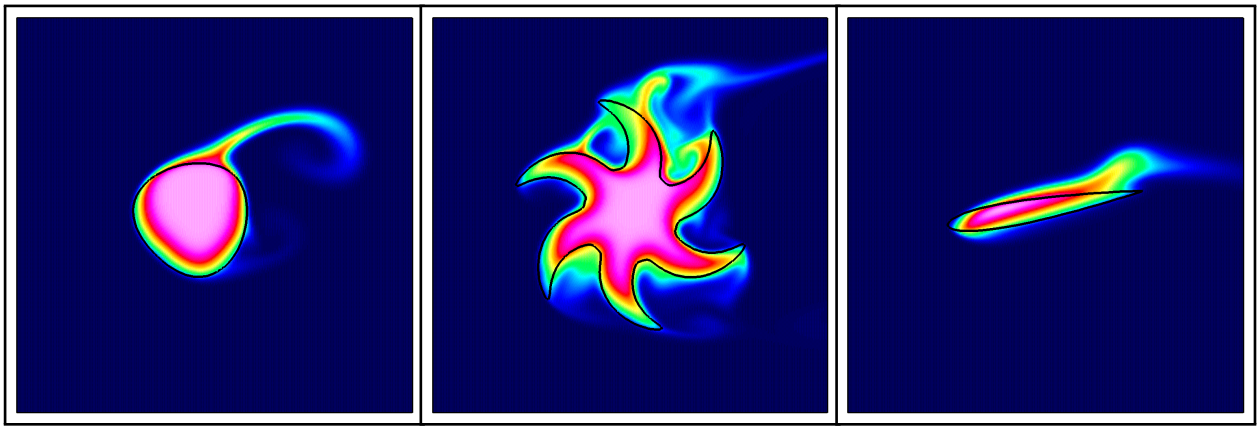

Figure 3: Examples of 2D body force regions defined from a curve used as immersed boundaries and heat sources. Left: buoyant incompressible flow past a heated *blob* - the curve was defined with the `NurbsMapping`. Middle: buoyant incompressible flow past a heated *starfish* - the curve was defined with the `NurbsMapping`. Right: buoyant incompressible flow past a heated *NACA airfoil* - the curve was defined with the `AirfoilMapping`.

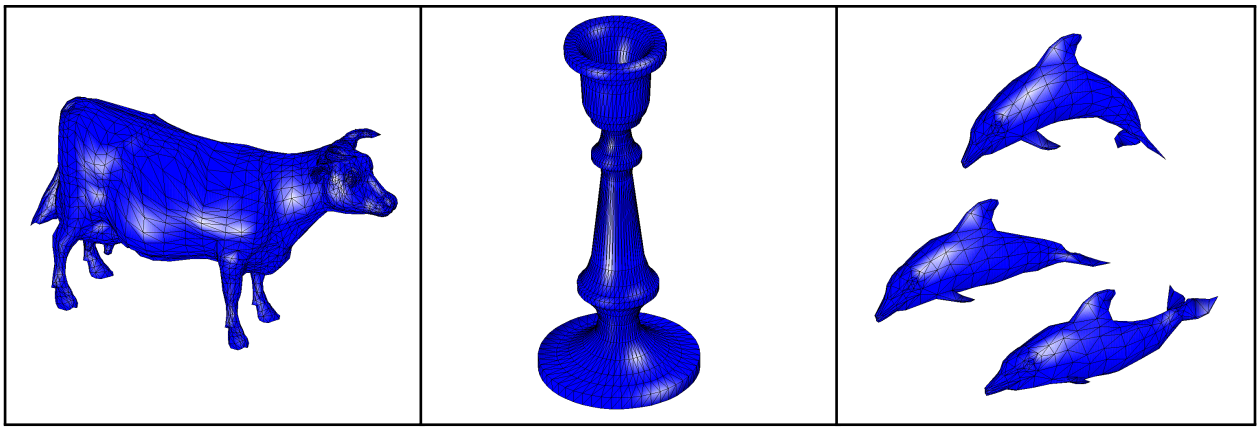

Figure 4: Examples of unstructured surfaces create from PLY files. These can be used as immersed boundary regions and heat sources in a Cgins computation.

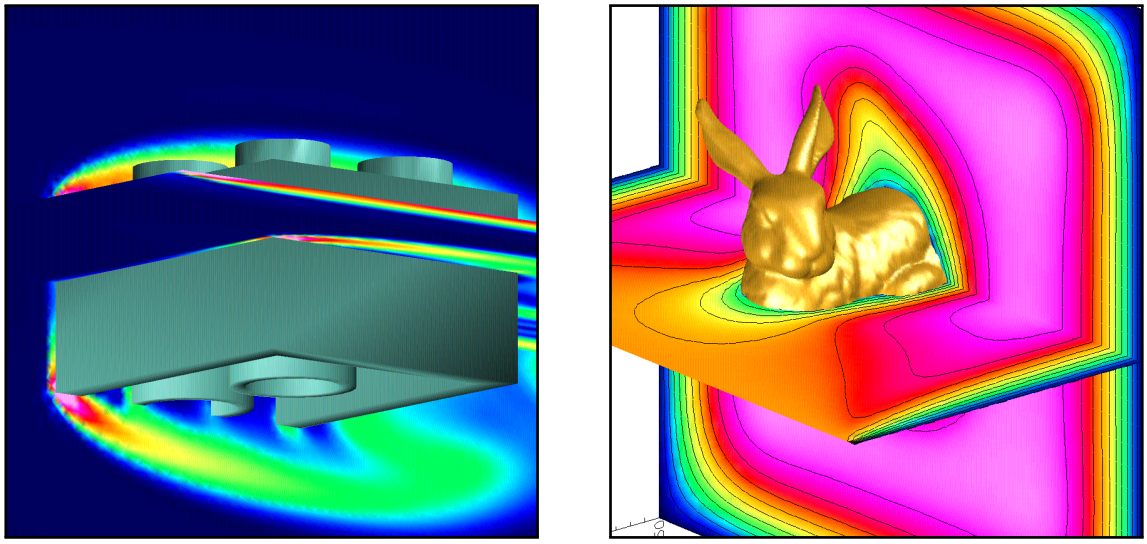

Figure 5: Examples of 3D body force regions defined from a unstructured surface used as immersed boundaries and heat sources. The UnstructuredMapping class is used to define the triangulated surface. Left: incompressible flow past a *LEGO* block - the surface was defined from an STL file. Right: incompressible flow past the the Stanford *bunny* - the surface was defined from a PLY file.

9.2 Defining body forces

For the incompressible Navier-Stokes (with Boussinesq approximation), the body forcings, $\mathbf{F_u}$ and F_T are added to the momentum and temperature equations as

$$\mathbf{u}_t + \mathbf{u}\cdot\nabla\mathbf{u} + \nabla p = \nu\Delta\mathbf{u} + \alpha\mathbf{g}(T - T_0) + \mathbf{F_u}, \tag{22}$$

$$T_t + \mathbf{u}\cdot\nabla T = \kappa\Delta T + F_T. \tag{23}$$

Drag force: The drag force consists of the sum of a linear and quadratic drag law,

$$\mathbf{F_u} = -\Big\{\beta_1(\mathbf{u}-\mathbf{u}_d) + -\beta_2|\mathbf{u}-\mathbf{u}_d|(\mathbf{u}-\mathbf{u}_d)\Big\}\mathcal{P}(\mathbf{x},t), \tag{24}$$

$$F_T = -\frac{\beta_T}{\Delta t}\mathcal{P}(\mathbf{x},t)(T - T_d), \tag{25}$$

where $\mathcal{P}(\mathbf{x},t)$ is a profile function. The *target* velocity is $\mathbf{u}_d$ and the target temperature is T_d. In general (to-do) the target functions can be a function of space and time $\mathbf{u}_d = \mathbf{u}_d(\mathbf{x},t)$, $T_d = T_d(\mathbf{x},t)$.

Immersed boundary: The *immersed boundary* force takes the form

$$\mathbf{F_u} = -\frac{\beta}{\Delta t}\mathcal{P}(\mathbf{x},t)(\mathbf{u}-\mathbf{u}_d), \tag{26}$$

$$F_T = -\frac{\beta_T}{\Delta t}\mathcal{P}(\mathbf{x},t)(T-T_d), \tag{27}$$

where $\mathcal{P}(\mathbf{x},t)$ is a profile function and Δt is the time-step.
Heat source: For the incompressible Navier-Stokes (with Boussinesq approximation) the heat source $F_T(\mathbf{x},t)$ is added to the right-hand-side of the temperature equation

$$F_T = \mathcal{P}(\mathbf{x},t)\frac{1}{C_p}F_T(\mathbf{x},t). \tag{28}$$

Question: should we scale the heat source by C_p ?

9.3 Defining boundary forcing

The *body-force regions* defined in Section 9.1 can be used to construct boundary conditions that vary over a a given boundary.

9.4 Defining body force profiles

9.4.1 Parabolic velocity profile

A 'parabolic' profile can be specified for a body or boundary forcing. The parabolic profile can be useful, for example, in specifying the velocity profile at an inflow boundary. The parabolic profile is zero at the boundary of the body force region and increases to a specified value $U_{\max}$ at a distance d from the boundary:

$$u(\mathbf{x}) = \begin{cases} U_{\max}(2-s/d)s/d & \text{if } s \le d \\ U_{\max} & \text{if } s > d \end{cases}$$

Here s is the shortest distance between a point $\mathbf{x}$ in the region to the boundary of the region, and d is the user specified *parabolic depth*.

10 Probes: defining output quantities at points and on regions

Probes can be used to output results during a simulation. A *point-probe* can be used, for example, to record the value of the solution at a given location over time. A *region-probe* can be used, for example, to record the integrated heat flux over a surface covered by multiple overlapping grids. There are five different types of probes:

grid point probe : output the solution at a given grid point over time.

location probe : output the solution at a given physical location over time.

region probe : output the value(s) (e.g. average value) of a quantity over a region (e.g. a portion of the domain or boundary contained within a specified box) over time.

boundary surface probe : output value(s) (e.g. integrated value) of a quantity over a specified boundary surface (e.g. surface of a body that is covered by multiple overlapping grids).

bounding box probe : output the values of the solution at all grid points that lie on the faces of a *index box* (the box being in the index space of a grid),

The values of probes can be output at a specified *frequency* (e.g. every 10 time-steps).

A probe is defined by its *type* (e.g. grid-point-probe, region-probe, ...) and its *quantity* (e.g. pressure, temperature, heat-flux). In addition, probes defined over regions or boundary surfaces are characterized by a *quantity-measure* (i.e. whether to save point-values, the average-value, or the total integrated (or summed) value), and by a *measure-type* (e.g. whether to use a discrete-sum or an integral to compute the average-value or total-value).

The probe output is saved to one or more output files (text files). Results from multiple region probes can be saved in the same file if desired. The results in these files can be plotted with the matlab scripts *plotProbes.m* for point-probes and *plotRegionProbe.m* for region-probes.

10.1 Point probes

Point probes are used to save solution values at specified positions within the domain.

The position of a *location* probe is specified by a physical location $\mathbf{x}$. The values at this location will be interpolated from the nearby values on the grid.

The position of a grid-point point probe can be specified by

1. inputing a grid point (i.e. by grid number and (i_1, i_2, i_3)).
2. specifying a grid point as the closest point to a given physical point $\mathbf{x}$ (the code will determine the nearest grid point).

10.2 Region probes

Examples of *region probes* are

1. output the average value of the velocity over a specified box enclosed in the domain,
2. output the integral of the temperature over the entire domain,
3. output the integral of the fluid force over the face of a body covered by multiple overlapping grids,
4. output the average heat flux over a portion of a boundary that lies within a specified box.

The region probe is defined by a *region*, a *quantity*, a *measure-type* and a *quantity-measure*.

Regions: Each *region probe* is associated with a region (volume or surface) that can be specified as one of the following types:

full domain region : choose the full domain,

box region : specify a box in physical space (that intersects the grid volume),

boundary region : choose a surface that is some portion of the boundary,

box boundary region : specify a box in physical space (that intersects grid boundaries),

The region can be chosen by specifying the corners of a box. It can also be specified by choosing an existing **body force region** (see Section 9), and using the region associated with that object.
The *surface* for a **boundary region** can be specified as

1. a list of grid faces $(side, axis, grid)$,
2. all grid faces with a specified *shared boundary flag* value,
3. choosing an existing **boundary force region** (see Section 9), and using the region associated with that object.

Quantity: The probe *quantity* defines the solution component(s) or derived quantity to be saved,

all-components : normally all components of the solution are saved for point probes,

density, velocity, temperature, pressure, .. : save any component of the solution (the available components depend on the equations being solved),

heat flux : $k\partial T/\partial n$

lift, drag, force : finish me ...

Measure-type: Region probes generally save an integrated value (sum or integral) over the probe *region* and the *measure-type* specifies whether to save the sum of discrete values or the integral,

sum : sum discrete values,

volume weighted sum : (Vsum) sum discrete values weighted by cell *volume*. For boundary regions the *volume* weighting becomes the cell area (3D) or cell arclength (2D).

integral : integrate values.

Quantity measure: The integrated value of a region probe can be normalized by the number of values or by the volume (or surface area) of the region:

average : provide the average of the quantity, i.e. sum of values divided by the number of values (if measure-type is *sum*), or integral of values divide by area (if measure-type is *integral*),

total : provide the total (sum or integral).

Here are mathematical descriptions of some of the probe options (MT=measure-type, QM=quantity-measure, R=region),

$$q_p = \sum_{\mathbf{i}} \chi_R(\mathbf{x_i})\, q_{\mathbf{i}}\, \delta_{\mathbf{i}}, \qquad \text{MT=sum/Vsum, QM=total,} \tag{29}$$

$$q_p = \frac{1}{N} \sum_{\mathbf{i}} \chi_R(\mathbf{x_i})\; q_{\mathbf{i}}\, \delta_{\mathbf{i}}, \quad N = \sum_{\mathbf{i}} \chi_R(\mathbf{x_i})\, \delta_{\mathbf{i}}, \qquad \text{MT=sum/Vsum, QM=average,} \tag{30}$$

$$q_p = \int_{\partial R} q\; dS \qquad \text{MT=integral, QM=total, R=boundary,} \tag{31}$$

$$q_p = \frac{1}{A} \int_{\partial R} q(\mathbf{x})\; dS, \quad A = \int_{\partial R} 1\; dS \qquad \text{MT=integral, QM=average, R=boundary.} \tag{32}$$

$$q_p = \int_{R} q\; d\mathbf{x} \qquad \text{MT=integral, QM=total, R=volume,} \tag{33}$$

$$q_p = \frac{1}{V} \int_{R} q(\mathbf{x})\; d\mathbf{x}, \quad V = \int_{R} 1\; d\mathbf{x} \qquad \text{MT=integral, QM=average, R=volume.} \tag{34}$$

Here $q_{\mathbf{i}}$ or $q(\mathbf{x})$ is the quantity to be evaluated (e.g. temperature or heat flux) and q_p is the probe value. The sum or integral is over the portion of the domain defined by the choice of region. $\chi_R(\mathbf{x})$ is the *characteristic* function for the region R, being 1 inside the region and 0 outside,

$$\chi_R(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in R, \\ 0, & \mathbf{x} \notin R. \end{cases} \tag{35}$$

The value of $\delta_{\mathbf{i}}$ is equal to 1, the cell volume, cell area or cell arclength, depending on the measure-type, and number of space dimensions,

$$\delta_{\mathbf{i}} = \begin{cases} 1, & \text{MT=sum,} \\ \delta V_{\mathbf{i}}, & \text{MT=Vsum, R=volume,} \\ \delta S_{\mathbf{i}}, & \text{MT=Vsum, R=boundary.} \end{cases} \tag{36}$$

The integral is computed with the Integrate class [10].

10.3 Bounding box probes

The bounding box probe can be used to output the solution on the faces (and ghost points) of a bounding box (in the index space of a grid). These values can be used, for example, to provide time dependent boundary conditions for a refined calculation that only solves the equations within the bounding box.

The bounding box is specified by

1. a component grid number, $g_{\rm bb}$,
2. an index box: [i1a,i1b], [i2a,i2b], [i3a,i3b] (in the index space of grid $g_{\rm bb}$),
3. number of layers n_l, (i.e. save this many lines of data on each face). For example, on the left face save the points $i_1 = i1a - n_l + 1, \ldots, i1a$.

11 Probe examples

In this section we provide various examples of using probes to query the simulation.

11.1 Temperature and heat flux probes

11.1.1 Heated Square or Box

We consider heat square transfer in the 2D unit square $\Omega = [0,1]^2$ or 3D unit box, $\Omega = [0,1]^3$. The Cgins command file `heatFlux.cmd` can be used to run various cases.

2D strip: We solve on a strip (periodic in y),

$$T_t = \kappa T_{xx} + f(x), \qquad 0 < x < 1, \tag{37}$$

$$T(0,y,t) = 0, \quad T(1,y,t) = 0. \tag{38}$$

For $f = f_0$ the exact steady state solution, boundary flux and average temperature are

$$T = \frac{f_0}{2\kappa} x(1-x), \tag{39}$$

$$kT_n(0,y) = -kT_x(0,y) = -\frac{kf_0}{2\kappa}, \tag{40}$$

$$\bar{T} = \int_0^1 \int_0^1 T(x,y)\,dxdy = -\frac{kf_0}{12\kappa} \tag{41}$$

12 Boundary conditions for the fourth-order method

Here are the analytic and numerical conditions that we impose at a boundary inorder to determine the values of $\mathbf{u}$ at the two ghost points.

noSlipWall: Analytic boundary conditions

$$\mathbf{u} = \mathbf{u}_B(\mathbf{x}, t)$$

plus numerical boundary conditions

$$\begin{aligned} \mathbf{t}_\mu \cdot \left\{ \nu \Delta \mathbf{u} - \nabla p - (\mathbf{u} \cdot \nabla)\mathbf{u} - \mathbf{u}_t \right\} &= 0 \\ \text{Extrapolate} \quad \mathbf{t}_\mu \cdot \mathbf{u} &= 0 \\ \nabla \cdot \mathbf{u} &= 0 \\ \partial_n (\nabla \cdot \mathbf{u}) &= 0 \end{aligned}$$

inflowWithVelocityGiven or outflow: Analytic boundary conditions for inflow are

$$\mathbf{u} = \mathbf{u}_I(\mathbf{x}, t) \qquad \text{(inflow)}$$

For outflow the equation is used on the boundary. The numerical boundary conditions are

$$\begin{aligned} \mathbf{t}_\mu \cdot (\mathbf{u}_{nn}) &= 0 \\ \text{Extrapolate} \quad \mathbf{t}_\mu \cdot \mathbf{u} &= 0 \\ \nabla \cdot \mathbf{u} &= 0 \\ \partial_n (\nabla \cdot \mathbf{u}) &= 0 \end{aligned}$$

slipWall: Analytic boundary conditions are

$$\mathbf{n} \cdot \mathbf{u} = \mathbf{n} \cdot \mathbf{u}_B$$

The numerical boundary conditions are

$$\begin{aligned} \mathbf{t}_\mu \cdot \left\{ \nu \Delta \mathbf{u} - \nabla p - (\mathbf{u} \cdot \nabla)\mathbf{u} - \mathbf{u}_t \right\} &= 0 \quad \text{determines } \mathbf{t}_\mu \cdot \mathbf{u} \text{ on the boundary} \\ \mathbf{t}_\mu \cdot (\mathbf{u}_n) &= 0 \\ \mathbf{t}_\mu \cdot (\mathbf{u}_{nnn}) &= 0 \\ \nabla \cdot \mathbf{u} &= 0 \\ \partial_n (\nabla \cdot \mathbf{u}) &= 0 \end{aligned}$$

Discretizing the Boundary conditions: For the purposes of this discussion assume that the boundary condition for $\mathbf{u}$ is of the form $\mathbf{u}(\mathbf{x}, t) = \mathbf{u}_B(\mathbf{x}, t)$ for $\mathbf{x} \in \partial\Omega$. More general boundary conditions on $\mathbf{u}$ and p, such as extrapolation conditions, can also be dealt with although some of the details of implementation may vary. At a boundary the following conditions are applied

$$\left.\begin{aligned} \mathbf{U}_i - \mathbf{u}_B(\mathbf{x}_i) &= 0 \\ \nabla_4 \cdot \mathbf{U}_i &= 0 \\ D_{4n}(\nabla_4 \cdot \mathbf{U}_i) &= 0 \\ \tfrac{d}{dt}\mathbf{U}_i + (\mathbf{U}_i \cdot \nabla_4)\mathbf{U}_i + \nabla_4 P_i - \nu \Delta_4 \mathbf{U}_i - \mathbf{f}_i &= 0 \\ \Delta_4 P_i + \sum_{m=1}^{n_d} \nabla_4 U_{m,i} \cdot D_{4x_m} \mathbf{U}_i - \nabla_4 \cdot \mathbf{f}_i &= 0 \end{aligned}\right\} \quad \text{for } i \in \text{Boundary}$$

$$\left.\begin{aligned} \mathbf{t}_\mu \cdot D^4_{+m} \mathbf{U}_i &= 0 \\ D^4_{+m} P_i &= 0 \end{aligned}\right\} \quad \text{for } i \in \text{2nd fictitious line}$$

where $\mathbf{t}_\mu$, $\mu = 1, n_d - 1$ are linearly independent vectors that are tangent to the boundary. In the extrapolation conditions either D_{+m} or D_{-m} should be chosen, as appropriate. Thus at each point along the boundary there are

12 equations for the 12 unknowns $(\mathbf{U}_i, P_i)$ located on the boundary and the 2 lines of fictitious points. Note that two of the numerical boundary conditions couple the pressure and velocity. In order to advance the velocity with an explicit time stepping method it convenient to decouple the solution of the pressure equation from the solution of the velocity. A procedure to accomplish this is described in the next section on time stepping.

Edges and Vertices: An important special case concerns obtaining solution values at points that lie near edges and vertices of grids (or corners of grids in 2D). Define a *boundary edge* to be the edge that is formed at the intersection of adjacent faces of the unit cube where both faces are boundaries of the computational domain. Along a boundary edge, values of the solution are required at the fictitious points in the region exterior to both boundary faces. For example, suppose that the edge defined by $i_1 = n_{1,a}$, $i_2 = n_{2,a}$ and $i_3 = n_{3,a}, \ldots, n_{3,b}$ is a boundary edge. Values must be determined at the exterior points $i = (n_{1,a} + m, n_{2,a} + n, i_3)$ for $m, n = -2, -1$.

Here we derive a more accurate formula than was in my paper. These expressions will be exact for polynomials of degree 4. By Taylor series,

$$u(r_1, r_2) = u(0,0) + \mathcal{D}_1(r_1, r_2) + \mathcal{D}_2(r_1, r_2) + \mathcal{D}_3(r_1, r_2) + \mathcal{D}_4(r_1, r_2) + O(|\mathbf{r}|^6)$$

where

$$\begin{aligned}
\mathcal{D}_1(r_1, r_2) &= (r_1 \partial_{r_1} + r_2 \partial_{r_2}) u(0,0) \\
\mathcal{D}_2(r_1, r_2) &= \frac{1}{2}({r_1}^2 \partial^2_{r_1} + {r_2}^2 \partial^2_{r_2} + 2 r_1 r_2 \partial_{r_1} \partial_{r_2}) u(0,0) \\
\mathcal{D}_3(r_1, r_2) &= \frac{1}{3!}({r_1}^3 \partial^3_{r} + {r_2}^3 \partial^3_{r_2} + 3{r_1}^2 r_2 \partial^2_{r_1} \partial_{r_2} + 3 r_1 {r_2}^2 \partial_{r_1} \partial^2_{r_2}) u(0,0)
\end{aligned}$$

We also have

$$u(-r_1, -r_2) = 2u(0,0) - u(r_1, r_2) + 2\mathcal{D}_2(r_1, r_2) + 2\mathcal{D}_4(r_1, r_2) + O(|\mathbf{r}|^6) \tag{42}$$

$$u(2r_1, 2r_2) = u(0,0) + 2\mathcal{D}_1(r_1, r_2) + 4\mathcal{D}_2(r_1, r_2) + 8\mathcal{D}_3(r_1, r_2) + 16\mathcal{D}_4(r_1, r_2) + O(|\mathbf{r}|^6) \tag{43}$$

$$8u(r_1, r_2) - u(2r_1, 2r_2) = 7u(0,0) + 6\mathcal{D}_1(r_1, r_2) + 4\mathcal{D}_2(r_1, r_2) - 8\mathcal{D}_4(r_1, r_2) + O(|\mathbf{r}|^6) \tag{44}$$

From equation (44) we can solve for $\mathcal{D}_4(r_1, r_2)$,

$$\mathcal{D}_4(r_1, r_2) = \frac{7}{8}u(0,0) - u(r_1, r_2) + \frac{1}{8}u(2r_1, 2r_2) + \frac{3}{4}\mathcal{D}_1(r_1, r_2) + \frac{1}{2}\mathcal{D}_2(r_1, r_2) + O(|r|^5 + |s|^5)$$

and substitute into equation (42)

$$u(-r_1, -r_2) = \frac{15}{4}u(0,0) - 3u(r_1, r_2) + \frac{1}{4}u(2r_1, 2r_2) + \frac{3}{2}\mathcal{D}_1(r_1, r_2) + 3\mathcal{D}_2(r_1, r_2) + O(|\mathbf{r}|^6) \tag{45}$$

We will use this last equation to determine u at the first corner ghost point, $\mathbf{U}_{-1,-1,i_3}$. Proceeding in a similar way it follows that

$$u(-2r_1, -r_2) = \frac{15}{4}u(0,0) - 3u(2r_1, r_2) + \frac{1}{4}u(4r_1, 2r_2) + \frac{3}{2}\mathcal{D}_1(2r_1, r_2) + 3\mathcal{D}_2(2r_1, r_2) + O(|\mathbf{r}|^6)O(|\mathbf{r}|^6) \tag{46}$$

$$u(-r_1, -2r_2) = \frac{15}{4}u(0,0) - 3u(r_1, 2r_2) + \frac{1}{4}u(2r_1, 4r_2) + \frac{3}{2}\mathcal{D}_1(r_1, 2r_2) + 3\mathcal{D}_2(r_1, 2r_2) + O(|\mathbf{r}|^6)O(|\mathbf{r}|^6) \tag{47}$$

$$u(-2r_1, -2r_2) = 30u(0,0) - 32u(r_1, r_2) + 3u(2r_1, 2r_2) + 24\mathcal{D}_1(r_1, r_2) + 24\mathcal{D}_2(r_1, r_2) + O(|\mathbf{r}|^6)O(|\mathbf{r}|^6) \tag{48}$$

from which we will determine the ghost points values at $\mathbf{U}_{-2,-2,i_3}$, $\mathbf{U}_{-2,-1,i_3}$ and $\mathbf{U}_{-1,-2,i_3}$. By symmetry we obtain formulae for ghost points outside all other edges in three-dimensions, $\mathbf{U}_{-2:-1,i_2,-2:-1}$, $\mathbf{U}_{i_1,-2:-1,-2:-1}$. For ghost points outside the vertices in three-dimensions we have

$$u(-r_1, -r_2, -r_3) = \frac{15}{4}u(0,0) - 3u(r_1, r_2, r_3) + \frac{1}{4}u(2r_1, 2r_2, 2r_3) \tag{49}$$

$$+ \frac{3}{2}\mathcal{D}_1(r_1, r_2, r_3) + 3\mathcal{D}_2(r_1, r_2, r_3) + O(|\mathbf{r}|^6)O(|\mathbf{r}|^6) \tag{50}$$

where

$$\mathcal{D}_1(r_1,r_2,r_3) = \Big(r_1\partial_{r_1} + r_2\partial_{r_2} + r_3\partial_{r_3}\Big)u(0,0,0)$$
$$\mathcal{D}_2(r_1,r_2,r_3) = \frac{1}{2}\Big({r_1}^2\partial^2_{r_1} + {r_2}^2\partial^2_{r_2} + {r_3}^2\partial^2_{r_3} + 2r_1r_2\partial_{r_1}\partial_{r_2} + 2r_1r_3\partial_{r_1}\partial_{r_3} + 2r_2r_3\partial_{r_2}\partial_{r_3}\Big)u(0,0,0)$$
$$\mathcal{D}_3(r_1,r_2,r_3) = \frac{1}{3!}\sum_{m_1=1}^{3}\sum_{m_2=1}^{3}\sum_{m_3=1}^{3} r_{m_1}r_{m_2}r_{m_3}\partial_{m_1}\partial_{m_2}\partial_{m_3}u(0,0,0)$$
$$= \frac{1}{3!}\Big({r_1}^3\partial^3_{r} + {r_2}^3\partial^3_{r_2} + {r_3}^3\partial^3_{r_3} + 3{r_1}^2r_2\partial^2_{r_1}\partial_{r_2} + 3r_1{r_2}^2\partial_{r_1}\partial^2_{r_2}$$
$$+ 3{r_1}^2r_3\partial^2_{r_1}\partial_{r_3} + 3r_1{r_3}^2\partial_{r_1}\partial^2_{r_3} + 3{r_2}^2r_3\partial^2_{r_2}\partial_{r_3} + 3r_2{r_3}^2\partial_{r_2}\partial^2_{r_3} + 6r_1r_2r_3\partial_{r_1}\partial_{r_2}\partial_{r_3}\Big)u(0,0,0)$$

In order to evaluate the formulae (45,46,47, 48,50) we need to evaluate the derivatives appearing in $\mathcal{D}_1$ and $\mathcal{D}_2$. All the non-mixed derivatives $\partial^m u(0,0)/\partial^m_{r_n}$, $m = 1, 2$, can be evaluated using the boundary values since these are all tangential derivatives. The second-order mixed derivative term such as $u_{r_1r_2}$ requires a bit more work. In two-dimensions we evaluate this term by taking the parametric derivatives of the divergence, $\partial_{r_m}\nabla\cdot\mathbf{u} = 0$. Since

$$\nabla\cdot\mathbf{u} = \sum_{n=1}^{2}(\partial_x r_n)\ u_{r_n} \ + \ \sum_{n=1}^{2}(\partial_y r_n)\ v_{r_n}$$

then $\partial_{r_m}\nabla\cdot\mathbf{u} = 0$ gives

$$\begin{aligned}(\partial_x r_2)\ u_{r_1r_2} + (\partial_y r_2)\ v_{r_1r_2} =& (\partial_x r_2)_{r_1}\ u_{r_2} + (\partial_x r_1)\ u_{r_1r_1} + (\partial_x r_1)_{r_1}\ \partial_{r_1}u + \\ & (\partial_y r_2)_{r_1}\ v_{r_2} + (\partial_y r_1)\ v_{r_1r_1} + (\partial_y r_1)_{r_1}\ \partial_{r_1}v \\ (\partial_x r_1)\ u_{r_1r_2} + (\partial_y r_1)\ v_{r_1r_2} =& (\partial_x r_2)\ u_{r_2r_2} + (\partial_x r_2)_{r_2}\ \partial_{r_2}u ...\end{aligned}$$

These last two equations can be solved for $u_{r_1r_2}$ and $v_{r_1r_2}$ in terms of known tangential derivatives.

In three dimensions taking the two parametric derivatives of the divergence,

$$(\partial_x r_2)\ u_{r_1r_2} + (\partial_y r_2)\ v_{r_1r_2} + (\partial_z r_2)\ w_{r_1r_2} = ... \tag{51}$$
$$(\partial_x r_1)\ u_{r_1r_2} + (\partial_y r_1)\ v_{r_1r_2} + (\partial_z r_1)\ w_{r_1r_2} = ... \tag{52}$$

gives only two equations for the three unknowns, $u_{r_1r_2}, v_{r_1r_2}$ and $w_{r_1r_2}$. We therefore add an extra condition by extrapolating the tangential component of the velocity,

$$\mathbf{t}_3\cdot D^6_{+,1,2}\mathbf{U}_{i_1-1,i_2-1,i_3} = 0 \tag{53}$$

Solve the last equation for $\mathbf{t}_3\cdot\mathbf{U}_{i_1-1,i_2-1,i_3}$ gives

$$\mathbf{t}_3\cdot\mathbf{U}_{i_1-1,i_2-1,i_3} = \mathcal{E}^6_{+,1,2}\mathbf{t}_3\cdot\mathbf{U}_{i_1-1,i_2-1,i_3} \tag{54}$$

where we have introduced the operator $\mathcal{E}^6_{+,1,2}$. By substituting this last equation (54) into $\mathbf{t}_3\cdot$ equation (45),

$$\mathbf{t}_3\cdot\mathbf{u}(-r_1,-r_2) = \mathbf{t}_3\cdot\left(\frac{15}{4}\mathbf{u}(0,0) - 3\mathbf{u}(r_1,r_2) + \frac{1}{4}\mathbf{u}(2r_1,2r_2) + \frac{3}{2}\mathcal{D}_1(r_1,r_2)\mathbf{u}(0) + 3\mathcal{D}_2(r_1,r_2)\mathbf{u}(0)\right) + O(|\mathbf{r}|^6)$$

we can eliminate $\mathbf{t}_3\cdot\mathbf{U}_{i_1-1,i_2-1,i_3}$ and obtain an equation for $\mathbf{t}_3\cdot\mathbf{u}_{r_1r_2}$ in terms of known quanitites,

$$\begin{aligned}\mathbf{t}_3\cdot\left(\mathcal{E}^6_{+,1,2}\mathbf{U}_{i_1-1,i_2-1,i_3}\right) = \mathbf{t}_3\cdot\Big(&\frac{15}{4}\mathbf{u}(0,0) - 3\mathbf{u}(r_1,r_2) + \frac{1}{4}\mathbf{u}(2r_1,2r_2) + \frac{3}{2}\mathcal{D}_1(r_1,r_2)\mathbf{u}(0) \\ &+ \frac{3}{2}({r_1}^2\partial^2_{r_1} + {r_2}^2\partial^2_{r_2} + {r_3}^2\partial^2_{r_3} + 2r_1r_2\partial_{r_1}\partial_{r_2} + 2r_1r_3\partial_{r_1}\partial_{r_3} + 2r_2r_3\partial_{r_2}\partial_{r_3})\mathbf{u}(0,0)\Big)\end{aligned}$$

or re-written as

$$\begin{aligned}\mathbf{t}_3\cdot\mathbf{u}_{r_1r_2}(0,0) = \frac{1}{3}\Big\{&\mathbf{t}_3\cdot\left(\mathcal{E}^6_{+,1,2}\mathbf{U}_{i_1-1,i_2-1,i_3}\right) - \mathbf{t}_3\cdot\Big(\frac{15}{4}\mathbf{u}(0,0) - 3\mathbf{u}(r_1,r_2) + \frac{1}{4}\mathbf{u}(2r_1,2r_2) + \frac{3}{2}\mathcal{D}_1(r_1,r_2)\mathbf{u}(0) \\ &+ \frac{3}{2}({r_1}^2\partial^2_{r_1} + {r_2}^2\partial^2_{r_2} + {r_3}^2\partial^2_{r_3} + 2r_1r_3\partial_{r_1}\partial_{r_3} + 2r_2r_3\partial_{r_2}\partial_{r_3})\mathbf{u}(0,0)\Big\}\end{aligned} \tag{55}$$

To summarize we solve equations (51,52,55) for the three unknowns $u_{r_1r_2}, v_{r_1r_2}$ and $w_{r_1r_2}$.

Solving the numerical boundary equations: The numerical boundary conditions (**??**) define the values of $\mathbf{U}$ on two lines of fictitious points in terms of values of the velocity on the boundary and the interior. The equations couple the unknowns in the tangential direction to the boundary so that in principle a system of equations for all boundary points must be solved. However, when the grid is nearly orthogonal to the boundary there is a much more efficient way to solve the boundary conditions. The first step in the algorithm is to solve for the tangential components of the velocity from

$$\left.\begin{array}{rcl} \mathbf{U}_i(t) - \mathbf{u}_B(\mathbf{x}_i,t) &=& 0 \\ \mathbf{t}_\mu \cdot \left\{ \frac{d}{dt}\mathbf{U}_i(t) + (\mathbf{U}_i(t)\cdot\nabla_4)\mathbf{U}_i(t) + \nabla_4 P^*(t) - \nu\Delta_4\mathbf{U}_i(t) - \mathbf{f} \right\} &=& 0 \end{array}\right\} \quad \text{for } i \in \text{Boundary}$$

$$\left.\begin{array}{rcl} \mathbf{t}_\mu \cdot D^6_{+m}(\mathbf{U}_i(t)) &=& 0 \end{array}\right\} \quad \text{for } i \in \text{Second fictitious line}$$

If the grid is orthogonal to the boundary then the discrete Laplacian applied at boundary will not have any mixed derivative terms. Therefore the only fictitious points appearing in the equation applied at the the boundary point (i_1,i_2,i_3) will be the two points (i_1,i_2,i_3-n) $n=1,2$ (here we assume that i_3 is in the normal direction to the boundary). Thus for each point on the boundary (i_1,i_2,i_3) the values of $\mathbf{t}_\mu\cdot\mathbf{u}$ can be determined at the fictitious points (i_1,i_2,i_3-1) and (i_1,i_2,i_3-2). There is no coupling between adjacent boundary points so no large system of equations need be solved. The tangential components of the velocity are determined for all fictitious points on the entire boundary. The second step is to determine the the normal component of the velocity at the fictitious points from

$$\left.\begin{array}{rcl} \mathbf{U}_i(t) - \mathbf{u}_B(\mathbf{x}_i,t) &=& 0 \\ \nabla_4\cdot\mathbf{U}_i(t) &=& 0 \\ D_{4n}(\nabla_4\cdot\mathbf{U}_i(t)) &=& 0 \end{array}\right\} \quad \text{for } i \in \text{Boundary}$$

If the grid is orthogonal to the boundary then the divergence on the boundary can be written in the form

$$\nabla\cdot\mathbf{u} = \frac{1}{e_1e_2e_3}\left\{ \frac{\partial}{\partial n}(e_2e_3\mathbf{n}\cdot\mathbf{u}) + \frac{\partial}{\partial t_1}(e_1e_3\mathbf{t}_1\cdot\mathbf{u}) + \frac{\partial}{\partial t_2}(e_1e_2\mathbf{t}_2\cdot\mathbf{u}) \right\}$$

where the e_m are functions of $\partial\mathbf{x}/\partial\mathbf{r}$. Note that only normal derivatives of $\mathbf{n}\cdot\mathbf{u}$ appear in the expression for the divergence. Thus, at a boundary point, (i_1,i_2,i_3), the stencil for $\nabla_4\cdot\mathbf{U}$ will only involve the fictitious points at (i_1,i_2,i_3-n), $n=1,2$. Similarly, the stencil for $D_{4n}(\nabla_4\cdot\mathbf{U})$ at a boundary will only involve the fictitious points at (i_1,i_2,i_3-n), $n=1,2$. Thus there is no coupling between adjacent boundary points and the unknown values for $\mathbf{n}\cdot\mathbf{u}$ can be easily determined. Note that the equations for $D_{4n}(\nabla_4\cdot\mathbf{U})$ will couple values for $\mathbf{t}_\mu\cdot\mathbf{u}$ at fictitious points along the boundary but these values have already been determined in the first step.

In practice the boundary conditions are solved in a correction mode – some initial guess is assumed for the values at the fictitious points and a correction is computed. If the grid is orthogonal or nearly orthogonal to the boundary then the first correction will give an accurate answer to the boundary conditions. If the grid is not orthogonal to the boundary then the solution procedure can repeated one or more times until a desired accuracy is achieved. This iteration should converge quickly provided that the grid is not overly skewed.

13 Turbulence models

The typical RANS model for the incompressible Navier-Stokes equations which uses the Boussinesq eddy viscosity approximation is

$$\begin{aligned}\partial_t u_i + u_k \partial_{x_k} u_i + \partial_{x_i} p &= \partial_{x_k}\Big((\nu + \nu_T)(\partial_{x_k} u_i + \partial_{x_i} u_k)\Big)\\ \tau_{i,j} &= (\nu + \nu_T)(\partial_{x_k} u_i + \partial_{x_i} u_k)\\ \widetilde{\nu} &= \nu + \nu_T\end{aligned}$$

where ν_T is the turbulent eddy viscosity.

13.1 Wall Shear Stress

The stress on a wall with normal $\mathbf{n}$ is

$$\boldsymbol{\tau}_{\text{wall}} = \boldsymbol{\tau} \cdot \mathbf{n}$$

The shear stress is the tangential component of this wall stress,

$$\begin{aligned}\boldsymbol{\tau}_{\text{shear}} &= \boldsymbol{\tau}_{\text{wall}} - (\mathbf{n} \cdot \boldsymbol{\tau}_{\text{wall}})\mathbf{n}\\ &= \tau_{i,j} n_j - (n_k \tau_{k,j} n_j) n_i\end{aligned}$$

In two-dimensions for a horizontal wall, $\mathbf{n} = [0, 1]^T$ and the wall shear stress is

$$\tau_w = (\nu + \nu_T)(\partial u/\partial y + \partial v/\partial x)$$

Question : Is ν_T on the wall equal to zero always?

In three-dimensions we define scalar wall shear stress τ_w in terms of the tangential component of the velocity at the wall, $\mathbf{u}_t$,

$$\begin{aligned}\tau_w &= \mathbf{t} \cdot \boldsymbol{\tau} \cdot \mathbf{n},\\ \mathbf{u}_{\tan} &= \mathbf{u} - (\mathbf{n} \cdot \mathbf{u})\mathbf{n},\\ \mathbf{t} &= \mathbf{u}_{\tan}/|\mathbf{u}_{\tan}|.\end{aligned}$$

Given the definition τ_w we can define (see for example Wilcox [17])

$u_\tau = \sqrt{\frac{\tau_w}{\rho}}$	Friction velocity
$\mathbf{u}^+ = \mathbf{u}/u_\tau$	non-dimensional velocity for near wall region
$y^+ = u_\tau y/\nu$	non-dimensional length for near wall region
$U = \lvert\mathbf{u}_{\tan}\rvert$	
$\frac{U}{u_\tau} = \frac{1}{\kappa}\ln\frac{u_\tau y}{\nu} + C$	law of the wall, κ=Kármán's constant
$U^+ = \frac{U}{u_\tau}$	non-dimensional velocity for near wall region
$U^+ = \frac{1}{\kappa}\ln(Ey^+)$	law of the wall, E=surface roughness parameter

For a horizontal wall, with flow in the x-direction, and $\rho = 1$, and assuming $\nu_T(0) = 0$, we have

$$\begin{aligned}u_\tau &= \sqrt{\nu U_y(0)}\\ y^+ &= y\sqrt{U_y(0)/\nu}\\ U(y) &= \frac{\sqrt{\nu U_y(0)}}{\kappa}\ln\left[Ey\sqrt{U_y(0)/\nu}\right]\end{aligned}$$

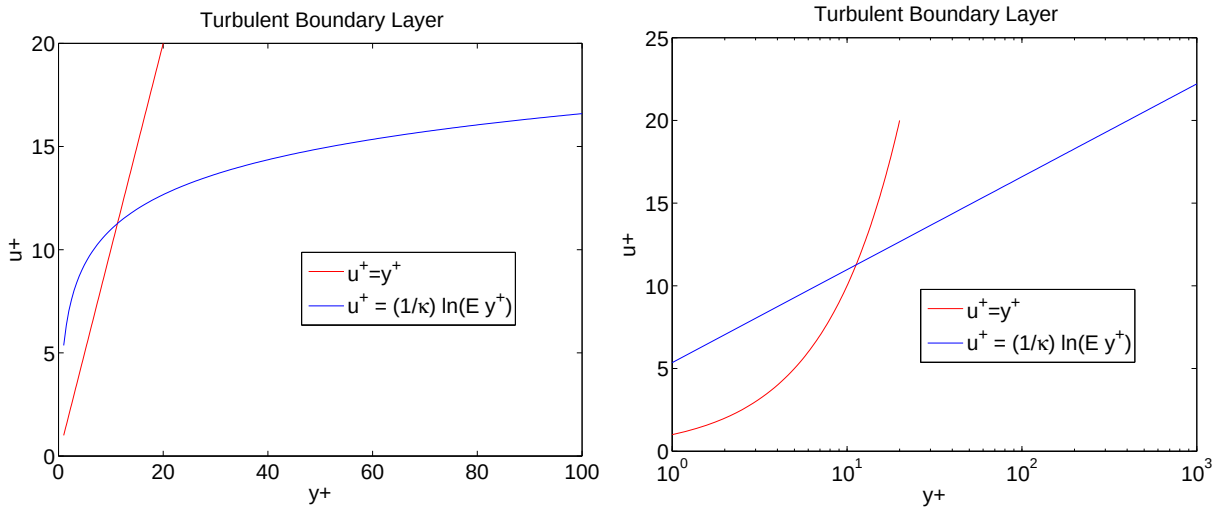

Figure 6: A turbulent boundary layer is thought to have an inner viscous sublayer where $u^+ = y+$ and an outer log-layer where $u^+ = \kappa^{-1}\ln(Ey^+)$.

13.2 Wall Functions

Wall functions are used to apply approximate boundary conditions in numerical simulations. They are used when the grid does not resolve the boundary layer.

The basic wall function approach assumes that the first grid point off the wall is located in the region where the law of the wall is valid. If y_p denotes the first grid point next to the wall then the law of the wall relates the tangential component of the velocity in the direction of the flow, $U(y_p)$, to the wall shear stress,

$$U(x,y_p) = \frac{u_\tau}{\kappa}\ln\left[Ey_p u_\tau/\nu\right] \tag{56}$$

$$U^+ = \kappa^{-1}\ln(Ey^+) \tag{57}$$

$$u_\tau = \sqrt{\nu u_y(x,0)} = \sqrt{\tau_w/\rho} \quad \text{(friction velocity)} \tag{58}$$

Here $E = 9$ for smooth walls and $\kappa = 0.41$. The law of the wall is thought to be valid in an attached turbulent boundary layer in region $\Omega_{\text{wall-law}} = \{\ \mathbf{x}\ |20 \le y^+ \le 100\}$. This condition replaces the wall no-slip boundary condition, $U(0) = 0$, on the component of the velocity in the direction of the flow (i.e. $U = \mathbf{t}\cdot\mathbf{U}$ where $\mathbf{t} = \mathbf{U}/|\mathbf{U}|$) . The condition is combined with the BC $\mathbf{n}\cdot\mathbf{U}(0) = 0$. In 3D we also need a condition on the other tangential component, which I think that we set to zero.

Note that $\lim_{y\to 0} U_y(x,y) \neq u_y(x,0)$ since the law of the wall does not hold for $y^+ < 20$. Thus the wall shear stress is defined using the fully resolved solution $u(x,y)$ and not the Reynold's averaged solution $U(x,y)$.

In the literature it is usually stated that the law of the wall is used to give τ_w, given a value $U(x,y_p)$, and this τ_w is used in the momentum equations to evaluate $\nabla\cdot(\boldsymbol{\tau})$ at the point y_p. This doesn't seem correct. The correct way would be to use the law of the way to evaluate $U_y(x,y)$ for a point near the wall and use this value in the momentum equation.

By differentiating the law of the wall w.r.t to y we get

$$U_y(x,y) = \frac{u_\tau}{\kappa y}\ ,\ U_{yy}(x,y) = -\frac{u_\tau}{\kappa y^2}, \tag{59}$$

which are valid for $\mathbf{x}\in\Omega_{\text{wall-law}}$.

So maybe we can implement the wall function by solving the full momentum equations on the first line in and by defining $U(x,0)$ from the two equations

$$\frac{U(x,y_p) - U(x,0)}{\Delta y} \approx U_y(x,y_p/2) \approx \frac{u_\tau}{\kappa}\frac{1}{y_p/2} \tag{60}$$

$$U(x,y_p) = \frac{u_\tau}{\kappa}\ln\left[Ey_p u_\tau/\nu\right] \tag{61}$$

The second equation gives u_τ as an implicit function of $U(x, y_p)$, $u_\tau = F(U(x, y_p))$

$$U(x, y_p) = \frac{u_\tau}{\kappa} \ln(E y_p u_\tau/\nu) \leftrightarrow u_\tau = F(U(x, y_p)) \tag{62}$$

$$\tag{63}$$

These equations can be thus used to eliminate u_τ to give $U(x, 0)$ in terms of $U(x, y_p)$, giving

$$U(x, 0) = U(x, y_p) - \frac{u_\tau}{\kappa} \frac{\Delta y}{y_p/2} \tag{64}$$

$$= U(x, y_p) - \frac{2}{\kappa} F(U(x, y_p)) \tag{65}$$

We can approximate this non-linear boundary condition by linearizing about some current guess for $U(x, y_p) \approx U_p^0$. Suppose that $F(U_p^0) = u_\tau^0$. Then expanding $F(U_p^0 + \delta U)$ gives

$$F(U_p^0 + \delta U) \approx F(U_p^0) + \frac{\partial F}{\partial U}(U_p^0)\delta U$$

Differentiating the expression (62) with respect to U gives (*check this*)

$$\frac{\partial F}{\partial U} = \frac{\partial u_\tau}{\partial U} = \frac{\kappa}{1 + \ln(E y_p u_\tau/\nu)} = \frac{\kappa}{1 + \kappa U/u_\tau}$$

Whence the linearized boundary condition is

$$\begin{aligned}
U(x, 0) &= U(x, y_p) - 2\kappa^{-1} F(U(x, y_p)) \\
&= U(x, y_p) - 2\kappa^{-1}\Big(F(U_p^0) + F_U(U_p^0)(U(x, y_p) - U_p^0)\Big) \\
&= \Big(1 - 2\kappa^{-1} F_U(U_p^0)\Big) U(x, y_p) - 2\kappa^{-1}\Big(F(U_p^0) - F_U(U_p^0)\ U_p^0\Big) \\
&= A_w U(x, y_p) + B_w \\
A_w(U_p^0) &= 1 - \frac{2}{1 + \kappa U_p^0/u_\tau^0} = \frac{\kappa U_p^0/u_\tau^0 - 1}{\kappa U_p^0/u_\tau^0 + 1} = \frac{\ln(E y_p^+) - 1}{\ln(E y_p^+) + 1} \\
B_w(U_p^0) &= -2\kappa^{-1} u_\tau^0 + \frac{2}{1 + \kappa U_p^0/u_\tau^0}\ U_p^0 = \frac{-2\kappa^{-1} u_\tau^0}{\kappa U_p^0/u_\tau^0 + 1}
\end{aligned}$$

This gives a boundary condition involving $U(x, 0)$ and $U(x, y_p)$ linearized about a state (U_p^0, u_τ^0) that satisfies the law of the wall. Also note that $0 < A_w < 1$ if $E y_p^+ > e$ (recall that $E = 9$ for smooth walls).

From Wilcox [17], the boundary conditions for k and ϵ for the $k - \epsilon$ model are then given by

$$k = \frac{u_\tau^2}{\sqrt{C_\mu}}, \qquad \epsilon = \frac{u_\tau^3}{\kappa y}$$

These could be imposed on the first line in, $y = y_p$?

Solving for u_τ from the law of the wall: Given a value $U_p = U(x, y_p)$ we need to be able to solve the nonlinear equation (62) to compute a value for $z = u_\tau$ satisfying

$$\begin{aligned}
G(z) &\equiv \kappa^{-1} z \ln(Az) - U_p = 0 \\
A &= E y_p/\nu \\
G'(z) &= \kappa^{-1}(1 + \ln(Az))
\end{aligned}$$

Note: If $U_p = 0$ then $z = 1/A$. Using Newton's method we get the iteration,

$$z^{k+1} = z^k - \frac{G(z^k)}{G'(z^k)}$$

There could be trouble with this iteration if $G'(z^k) = 0$, that is when $Az^k = e^{-1}$. However, if $U_p \geq 0$ then Az should satisfy $Az \geq 1 > e^{-1}$. This can seen from the fact that $z \ln(Az) \leq 0$ for $Az \leq 1$. Therefore $G(z) = 0$ at $z = 1/A$ and thus z should always satisfy $z > 1/A$ which imples $G'(z) \geq \kappa^{-1}$.

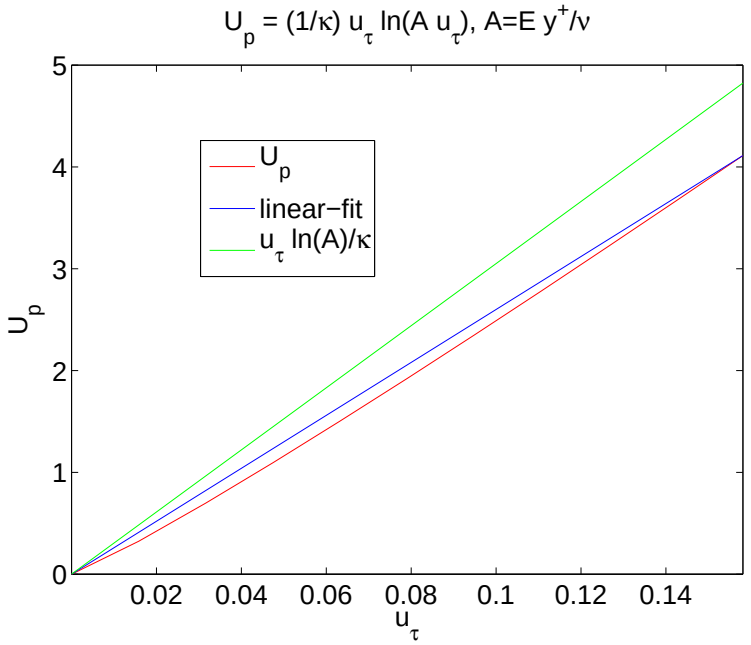

Figure 7: A plot of U_p versus u_τ for the law-of-the wall function, $U_p = \kappa^{-1} u_\tau \ln(A u_\tau)$.

An initial guess for z may come from the expression

$$z = \frac{\kappa U_p}{\ln(E y_p^+)}, \quad \rightarrow \quad \frac{\kappa U_p}{\ln(100E)} < z^0 < \frac{\kappa U_p}{\ln(20E)}$$

if we assume that $y_p^+ \in [20, 100]$.

In a boundary layer we might expect that as $R_e \to \infty$,

$$\tau_w \sim R_e^{-1/2} \ll 1$$
$$u_\tau \sim R_e^{-1/4} \ll 1$$

The viscous length scale is $\lambda_{\min} = \nu/u_\tau$. If the first grid line is at $y_p = M\lambda_{\min}$ (i.e. at $y^+ = M$) with $M \in [20, 100]$, then

$$A = E y_p/\nu = EM/u_\tau \sim EM R_e^{1/2} \gg 1$$
$$A u_\tau = EM$$

The product $EM \approx 9M$ is quite large.

Figure 7 shows a plot of U_p verses u_τ for the law-of-the wall function We see that a linear fit $u_\tau = aU_p + 1/A$ is a good approximation where the constant a is chosen as

$$a = \frac{u_\tau^m - 1/A}{\kappa^{-1} u_\tau^m \ln(A u_\tau^m)}$$

so that $U_p = \kappa^{-1} u_\tau^m \ln(A u_\tau^m)$ for some u_τ^m which represents some average value for the expected values for u_τ.

13.3 Baldwin-Lomax Zero Equation Model

The Baldwin-Lomax zero equation model is a two layer model [17]. The inner and outer layers are given by

$$\nu_T = \begin{cases} \nu_{T_i} = \rho l_{\text{mix}}^2 |\omega| & \text{if } y < y_c \\ \nu_{T_o} = \rho \alpha C_{cp} F_{\text{wake}} F_{\text{Kleb}}(y; y_{\max}/C_{\text{Kleb}}) & \text{if } y > y_c \end{cases},$$

where

$$\begin{aligned} y_c &= \min y \ \text{ such that } \ \nu_{T_i} = \nu_{T_o}, \\ l_{\text{mix}} &= \kappa y \Big[1 - e^{-y^+/A_0^+}\Big], \\ F_{\text{wake}} &= \min\left[y_{\max} F_{\max}; C_{wk} y_{\max} U_{\text{dif}}^2 / F_{\max}\right] \\ F_{\text{Kleb}}(y;\delta) &= [1 + 5.5(y/\delta)^6]^{-1} \qquad \text{(Klebanoff intermittency function)} \end{aligned}$$

and where $y_{\max}$ and $F_{\max}$ are determined from the maximum of the function

$$F(y) = y\omega\Big[1 - e^{-y^+/A_0^+}\Big]$$

and ω is the magnitude of the vorticity,

$$\begin{aligned} \omega^2 &= (v_x - u_y)^2 + (w_y - v_z)^2 + (u_z - w_x)^2 \\ &= 2\Omega_{ij}\Omega_{ij} \\ \Omega_{ij} &= \frac{1}{2}\Big(\partial u_i/\partial x_j - \partial u_j/\partial x_i\Big) \end{aligned}$$

For boundary layer flows U_{dif} is the maximum value of $\|\mathbf{u}\|$ through the layer?? For more general flows it is

$$U_{\text{dif}} = \Big(\|\mathbf{u}\|\Big)_{\max} - \Big(\|\mathbf{u}\|\Big)\Big|_{y=y_{\max}}$$

Note comment in Wilcox that U_{dif} is NOT the difference between the max and min velocities as specified in the original BL paper and at `http://www.cfd-online.com/Wiki/Baldwin-Lomax_model`

The model parameters are given by

$$\begin{aligned} \kappa = .40, \quad \alpha = 0.0168, \quad A_0^+ = 26 \\ C_{cp} = 1.6, \quad C_{\text{Kleb}} = 0.3 \quad C_{wk} = 1 \text{(or .25 from cfd-online)} \end{aligned}$$

13.4 Spalart-Allmaras turbulence model

Spalart-Allmaras **one equation model**

$$\begin{aligned} \nu_T &= \tilde{\nu} f_{v1} \\ \partial_t \tilde{\nu} + U_j \partial_j \tilde{\nu} &= c_{b1} \tilde{S} \tilde{\nu} - c_{w1} f_w (\tilde{\nu}/d)^2 + \frac{1}{\sigma}\Big[\partial_k[(\nu + \tilde{\nu})\partial_k \tilde{\nu}] + c_{b2} \partial_k \tilde{\nu} \partial_k \tilde{\nu}\Big] \\ c_{b1} = .1355, c_{b2} = .622, & c_{v1} = 7.1, \sigma = 2/3 \\ c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{(1 + c_{b2})}{\sigma}, \quad c_{w2} = 0.3, \quad c_{w3} = 2, \quad \kappa = .41 \\ f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \quad f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}}, f_w = g\left[\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6}\right]^{1/6} \\ \chi = \frac{\tilde{\nu}}{\nu}, \quad g = r + c_{w2}(r^6 - r), \qquad r = \frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2} \\ \tilde{S} = S + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}, \qquad S = \sqrt{2\Omega_{ij}\Omega_{ij}} \\ \Omega_{ij} = (1/2)(\partial_i U_j - \partial_j U_i) \qquad \text{rotation tensor} \end{aligned}$$

Depends on d, the distance to the nearest surface.

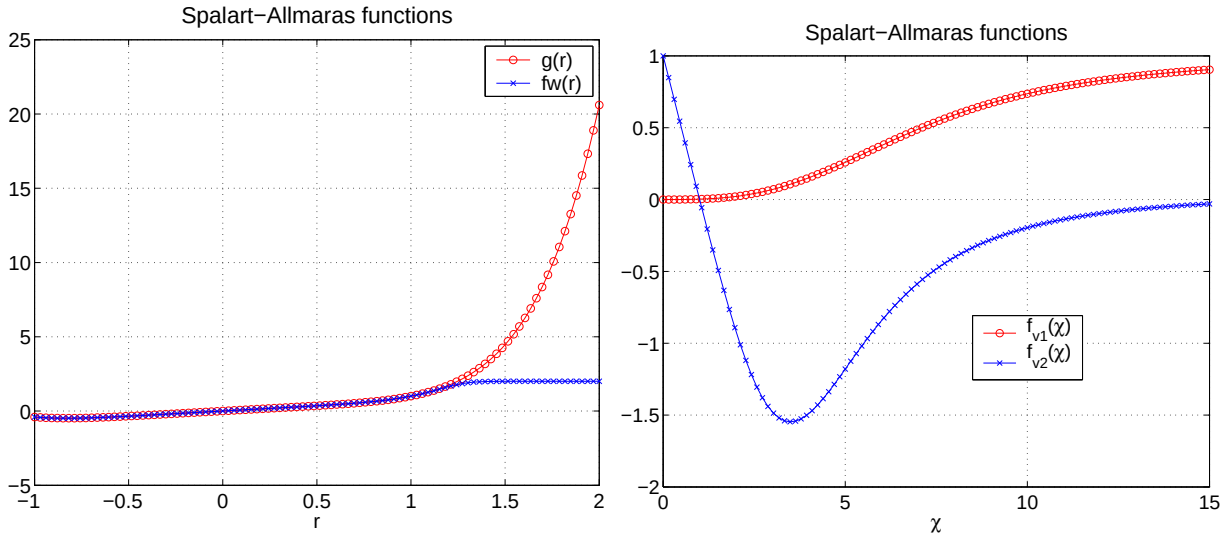

Figure 8: Behaviour of the SpalartConvergence fudge functions

Notes f_{v2} can be positive or negative but is bounded from above by 1 and below by ??

$$\begin{aligned} f_{v2} &= 1 - \frac{\chi}{1+\chi f_{v1}} \\ &= 1 - \frac{1}{\chi^{-1} + 1/[1+(c_{v1}/\chi)^{-3}]} \\ &\to 0 \qquad \text{as } \chi \to \infty \\ &\to 1 \qquad \text{as } \chi \to 0 \\ &\approx 1 - \frac{1}{(1/7)+(1/2)} = -3/4 \qquad \text{when } \chi = c_{v1} \end{aligned}$$

Since f_{v2} can be negative, so can r and g.

On a rectangular grid this is discretized as

$$\begin{aligned} \partial_t \tilde{\nu}_{\mathbf{i}} + U_{\mathbf{i}} D_{0x} U_{\mathbf{i}} + V_{\mathbf{i}} D_{0y} U_{\mathbf{i}} = c_{b1}\tilde{S}_{\mathbf{i}}\tilde{\nu}_{\mathbf{i}} - c_{w1} f_w (\tilde{\nu}/d)^2 \\ + \frac{1}{\sigma} D_{+x}[(\nu + \tilde{\nu}_{i_1-\frac{1}{2}}) D_{-x}\tilde{\nu}_{\mathbf{i}} + D_{+y}[(\nu + \tilde{\nu}_{i_2-\frac{1}{2}}) D_{-y}\tilde{\nu}_{\mathbf{i}}] \\ + c_{b2}\Big\{ (D_{0x}\tilde{\nu})^2 + (D_{0y}\tilde{\nu})^2 \Big\} \end{aligned}$$

On curvilinear grids we use the conservative form of the second order term

$$\nabla \cdot (a \nabla \phi) = \frac{1}{J}\Big\{ \frac{\partial}{\partial r_1}\left(A^{11}\frac{\partial \phi}{\partial r_1}\right) + \frac{\partial}{\partial r_2}\left(A^{22}\frac{\partial \phi}{\partial r_2}\right) + \frac{\partial}{\partial r_1}\left(A^{12}\frac{\partial \phi}{\partial r_2}\right) + \frac{\partial}{\partial r_2}\left(A^{21}\frac{\partial \phi}{\partial r_1}\right) \Big\}$$

where

$$\begin{aligned} A^{11} &= aJ\left[\frac{\partial r_1}{\partial x_1}^2 + \frac{\partial r_1}{\partial x_2}^2 \right] \\ A^{22} &= aJ\left[\frac{\partial r_2}{\partial x_1}^2 + \frac{\partial r_2}{\partial x_2}^2 \right] \\ A^{12} &= aJ\left[\frac{\partial r_1}{\partial x_1}\frac{\partial r_2}{\partial x_1} + \frac{\partial r_1}{\partial x_2}\frac{\partial r_2}{\partial x_2} \right] \end{aligned}$$

A **second-order accurate** compact discretization to this expression is

$$\nabla \cdot (a \nabla \phi) \approx \frac{1}{J}\Big\{ D_{+r_1}\left(A^{11}_{i_1-\frac{1}{2}} D_{-r_1}\phi\right) + D_{+r_2}\left(A^{22}_{i_2-\frac{1}{2}} D_{-r_2}\phi\right) + D_{0r_1}\left(A^{12} D_{0r_2}\phi\right) + D_{0r_2}\left(A^{21} D_{0r_1}\phi\right) \Big\}$$

where we can define the cell average values for A^{mn} by

$$A^{11}_{i_1-\frac{1}{2}} \approx \frac{1}{2}(A^{11}_{i_1} + A^{11}_{i_1-1})$$
$$A^{22}_{i_2-\frac{1}{2}} \approx \frac{1}{2}(A^{22}_{i_2} + A^{22}_{i_2-1})$$

13.5 $k-\epsilon$ turbulence model

Here is the $k-\epsilon$ model

$$\begin{aligned}
\nu_T &= C_\mu k^2/\epsilon \\
\partial_t k + U_j\partial_j k &= \tau_{ij}\partial_j U_i - \epsilon + \partial_j[(\nu+\nu_T/\sigma_k)\partial_j k] \\
\partial_t \epsilon + U_j\partial_j \epsilon &= C_{\epsilon 1}\frac{\epsilon}{k}\tau_{ij}\partial_j U_i - C_{\epsilon 2}\epsilon^2/k + \partial_j[(\nu+\nu_T/\sigma_\epsilon)\partial_j \epsilon] \\
C_{\epsilon 1}=1.44, \quad C_{\epsilon 2}=1.92, \quad C_\mu &= .09, \quad \sigma_k = 1, \quad \sigma_\epsilon = 1.3
\end{aligned}$$

The production term is $P = \tau_{ij}\partial_j U_i$

$$\begin{aligned}
P &= \tau_{ij}\partial_j U_i \\
&= \nu_T(\partial_j U_i + \partial_i U_j)\partial_j U_i \\
&= \frac{\nu_T}{2}(\partial_j U_i + \partial_i U_j)(\partial_j U_i + \partial_i U_j) \\
&= \frac{\nu_T}{2}\Big((2u_x)^2 + (2v_y)^2 + (2w_z)^2 + 2(u_y+v_x)^2 + 2(u_z+w_x)^2 + 2(v_z+w_y)^2\Big) \\
&= \nu_T\Big(2(u_x^2+v_y^2+w_z^2) + (u_y+v_x)^2 + (u_z+w_x)^2 + (v_z+w_y)^2\Big)
\end{aligned}$$

Note: Here are some references that may be useful: *On the implementation of the $k-\epsilon$ turbulence model in incompressible flow solvers based on a finite element discretization* by Kuzmin and Mierka (2006), and *A note on the numerical treatment of the k-epsilon turbulence model* by Lew, Buscaglia and Carrica (200?). The authors make some recommendations on how to solve the $k-\epsilon$ equations. We may want to adopt some of these suggestions.

13.5.1 Full implicit solution of the k-ϵ turbulence model

The k-ϵ turbulence model can be solved using a *full-implicit* approximation. This form is not the most memory efficient but should be more robust. In the full-implicit approximation, the variables $(\mathbf{u}, k, \epsilon)$ are solved using a fully coupled system, with the pressure being solved as a separate solve.

The implicit time-stepping procedure is described in Section 5.3. For the k-ϵ equations the implicit operator $L(\mathbf{u}, k, \epsilon; \mathbf{u}^*, k^*, \epsilon^*)$ is given for the $\mathbf{u}$, k and ϵ equations by

$$\begin{aligned}
L^{\mathbf{u}} &= (\mathbf{u}^*\cdot\nabla)\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}^* - \nabla(\nu_T^*\nabla\mathbf{u}), \\
L^{k} &= (\mathbf{u}^*\cdot\nabla)k + (\mathbf{u}\cdot\nabla)k^* - \nabla((\nu+\nu_T^*/\sigma_k)\nabla k + \epsilon - (2P^*/k^*)k + (P^*/\epsilon^*)\epsilon, \\
L^{\epsilon} &= (\mathbf{u}^*\cdot\nabla)\epsilon + (\mathbf{u}\cdot\nabla)\epsilon^* - \nabla((\nu+\nu_T^*/\sigma_\epsilon)\nabla\epsilon - C_{\epsilon 1}C_\mu S_0^* k + C_{\epsilon 2}(e^*/k^*)^2 k + C_{\epsilon 2}\frac{2\epsilon^*}{k^*}\epsilon,
\end{aligned}$$

where $\nu_T^* = C_\mu(k^*)^2/\epsilon^*$. Here the *star* states $\mathbf{u}^*$, k^*, and ϵ^* denote the states that are used to linearize the equations. A large sparse matrix is formed for the implicit system and this is kept fixed for some number of time steps (chosen by the `refactor frequency` option).

Notes:

1. The main implicit time-stepping routine is in `common/src/ims.C`. The RHS for the implicit solve is computed ins `ins/src/ins.C`.
2. The file `ins/src/insImpKE.bf` defines the implicit system that is solved for the k-ϵ equations.
3. The file `ins/src/insImp.h` defines some generic macros and subroutines that are used for different implicit solvers (including the k-ϵ and Baldwin-Lomax equations, `insImpBL.bf`).
4. The file `ins/src/turbulenceModels.C` defines the boundary conditions for the turbulence models used for explicit time stepping and currently these are also used for the implicit time steppers.
5. The fortran routines are called from the C++ file `ins/src/implicit.C` in function `insImplicitMatrix`.

13.6 Diffusion Operator

When a turbulence model is added to the incompressible Navier-Stokes equation the diffusion operator usually takes the form of

$$\mathcal{D}_i = \sum_j \partial_{x_j} \Big(\nu_T (\partial_{x_i} u_j + \partial_{x_j} u_i) \Big)$$

where we will write ν_T instead of $\nu + \nu_T$ in this section. In particular

$$\begin{aligned}
\mathcal{D}_u &= \partial_x(2\nu_T u_x) + \partial_y(\nu_T u_y) + \partial_z(\nu_T u_z) + \partial_y(\nu_T v_x) + \partial_z(\nu_T w_x) \\
\mathcal{D}_v &= \partial_x(\nu_T v_x) + \partial_y(2\nu_T v_y) + \partial_z(\nu_T v_z) + \partial_x(\nu_T u_y) + \partial_z(\nu_T w_y) \\
\mathcal{D}_w &= \partial_x(\nu_T w_x) + \partial_y(\nu_T w_y) + \partial_z(2\nu_T w_z) + \partial_y(\nu_T v_z) + \partial_x(\nu_T u_z)
\end{aligned}$$

We can write these in a more "symmetric" form as follows. Since $u_x + v_y + w_z = 0$, it follows that

$$\partial_x(\nu_T u_x) = -\partial_x(\nu_T v_y) - \partial_x(\nu_T w_z)$$

and thus

$$\mathcal{D}_u = \partial_x(\nu_T u_x) + \partial_y(\nu_T u_y) + \partial_z(\nu_T u_z) + \partial_y(\nu_T v_x) - \partial_x(\nu_T v_y) + \partial_z(\nu_T w_x) - \partial_x(\nu_T w_z)$$

Therefore the diffusion operator can be written in a form where the principle part is the same for all components,

$$\begin{aligned}
\mathcal{D}_u &= \nabla \cdot (\nu_T \nabla u) + \partial_y(\nu_T v_x) - \partial_x(\nu_T v_y) + \partial_z(\nu_T w_x) - \partial_x(\nu_T w_z) \\
\mathcal{D}_v &= \nabla \cdot (\nu_T \nabla v) + \partial_z(\nu_T w_y) - \partial_y(\nu_T w_z) + \partial_x(\nu_T u_y) - \partial_y(\nu_T u_x) \\
\mathcal{D}_w &= \nabla \cdot (\nu_T \nabla w) + \partial_x(\nu_T u_z) - \partial_z(\nu_T u_x) + \partial_y(\nu_T v_z) - \partial_z(\nu_T v_y)
\end{aligned}$$

Note that the highest order derivatives cancel in the last four terms in these expressions,

$$\begin{aligned}
\mathcal{D}_u &= \nabla \cdot (\nu_T \nabla u) + \partial_y(\nu_T) v_x - \partial_x(\nu_T) v_y + \partial_z(\nu_T) w_x - \partial_x(\nu_T) w_z \\
\mathcal{D}_v &= \nabla \cdot (\nu_T \nabla v) + \partial_z(\nu_T) w_y - \partial_y(\nu_T) w_z + \partial_x(\nu_T) u_y - \partial_y(\nu_T) u_x \\
\mathcal{D}_w &= \nabla \cdot (\nu_T \nabla w) + \partial_x(\nu_T) u_z - \partial_z(\nu_T) u_x + \partial_y(\nu_T) v_z - \partial_z(\nu_T) v_y
\end{aligned}$$

13.7 Revised pressure equation

When a turbulence model is added to the incompressible Navier-Stokes equation the diffusion operator usually takes the form of

$$\mathcal{D}_i = \sum_j \partial_{x_j} \Big(\nu_T (\partial_{x_i} u_j + \partial_{x_j} u_i) \Big)$$

The pressure equation is derived from taking the divergence of the momentum equations,

$$\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p = \mathcal{D}$$

and using $\nabla \cdot \mathbf{u} = 0$ to give

$$\Delta p = -\nabla \mathbf{u} : \nabla \mathbf{u} + \nabla \cdot \mathcal{D}$$

For a constant viscosity, the last term on the right hand side is zero. When the viscosity is not constant we need to include the divergence of the diffusion operator in the equation for the pressure. This takes the form

$$\begin{aligned}
\nabla \cdot \mathcal{D} &= \sum_i \sum_j \partial_{x_i} \partial_{x_j} \Big[\nu_T \Big(\partial_{x_j} u_i + \partial_{x_i} u_j \Big) \Big] \\
&= \sum_j \partial_{x_j} \Big[\sum_i \partial_{x_i} \nu_T \ \partial_{x_j} u_i \Big] + \sum_i \partial_{x_i} \Big[\sum_j \partial_{x_j} \nu_T \ \partial_{x_i} u_j \Big] \\
&= 2 \ \sum_i \partial_{x_i} \Big[\sum_j \ \partial_{x_j} \nu_T \ \partial_{x_i} u_j \Big]
\end{aligned}$$

where we have again used $\nabla \cdot \mathbf{u} = 0$.

In two dimensions this takes the form

$$\begin{aligned}\nabla \cdot \mathcal{D}^{(2\mathrm{d})} &= 2\,\Big[\partial_x\Big(\partial_x\nu_T\;\partial_x u + \partial_y\nu_T\;\partial_x v\Big) + \partial_y\Big(\partial_x\nu_T\;\partial_y u + \partial_y\nu_T\;\partial_y v\Big)\Big] \\ &= 2\,\Big[\partial_x\nu_T\;\Delta u + \partial_x^2\nu_T\;\partial_x u + \partial_x\partial_y\nu_T\;\partial_y u \\ &\quad + \partial_y\nu_T\;\Delta v + \partial_x\partial_y\nu_T\;\partial_x v + \partial_y^2\nu_T\;\partial_y v\Big]\end{aligned}$$

In three dimensions,

$$\begin{aligned}\nabla \cdot \mathcal{D}^{(3\mathrm{d})} &= 2\,\Big[\partial_x\Big(\partial_x\nu_T\;\partial_x u + \partial_y\nu_T\;\partial_x v + \partial_z\nu_T\;\partial_x w\Big) \\ &\quad + \partial_y\Big(\partial_x\nu_T\;\partial_y u + \partial_y\nu_T\;\partial_y v + \partial_z\nu_T\;\partial_y w\Big) \\ &\quad + \partial_z\Big(\partial_x\nu_T\;\partial_z u + \partial_y\nu_T\;\partial_z v + \partial_z\nu_T\;\partial_z w\Big)\Big] \\ &= 2\,\Big[\partial_x\nu_T\;\Delta u + \partial_x^2\nu_T\;\partial_x u + \partial_x\partial_y\nu_T\;\partial_y u + \partial_x\partial_z\nu_T\;\partial_z u \\ &\quad + \partial_y\nu_T\;\Delta v + \partial_x\partial_y\nu_T\;\partial_x v + \partial_y^2\nu_T\;\partial_y v + \partial_y\partial_z\nu_T\;\partial_z v \\ &\quad + \partial_z\nu_T\;\Delta w + \partial_x\partial_z\nu_T\;\partial_x w + \partial_y\partial_z\nu_T\;\partial_y w + \partial_z^2\nu_T\;\partial_z w\Big]\end{aligned}$$

The addition of artificial dissipation also changes the pressure equation. The second-order artificial dissipation is

$$\mathbf{d}_{2,i} = (\texttt{ad21} + \texttt{ad22}|\nabla_h \mathbf{V}_i|_1) \sum_{m=1}^{n_d} \Delta_{m+}\Delta_{m-}\mathbf{V}_i \tag{66}$$

while the the fourth-order one is

$$\mathbf{d}_{4,i} = -\,(\texttt{ad41} + \texttt{ad42}|\nabla_h \mathbf{V}_i|_1) \sum_{m=1}^{n_d} \Delta_{m+}^2\Delta_{m-}^2\mathbf{V}_i \tag{67}$$

The artificial dissipation is of the form

$$\begin{aligned}\mathbf{d} &= \Big[\alpha_0 + \alpha_1\mathcal{G}(\nabla\mathbf{u})\Big] \sum_{m=1}^{n_d} \Delta_{m+}^p\Delta_{m-}^p\mathbf{u} \\ \mathcal{G}(\nabla\mathbf{u}) &= |u_x| + |u_y| + |v_x| + |v_y| + \ldots\end{aligned}$$

Taking the divergence of this expression results in

$$\nabla \cdot \mathbf{d} = \alpha_1\Big[\mathcal{G}_x \sum_m \Delta_{m+}^p\Delta_{m-}^p U + \mathcal{G}_y \sum_m \Delta_{m+}^p\Delta_{m-}^p V\Big]$$

where

$$\mathcal{G}_x = \mathrm{sgn}(u_x)\;u_{xx} + \mathrm{sgn}(u_y)\;u_{xy} + \mathrm{sgn}(v_x)\;v_{xx} + \mathrm{sgn}(v_y)\;v_{xy} + \ldots$$

and $\mathrm{sgn}(x)$ is $+1$, -1 or 0 for $x > 0$, $x < 0$ or $x = 0$.

13.7.1 Revised pressure boundary condition

The *pressure boundary condition* also is changed to include the new diffusion operator:

$$\partial_n p = \mathbf{n} \cdot \Big\{ -\partial_t\mathbf{u} - (\mathbf{u}\cdot\nabla)\mathbf{u} + \mathcal{D}\Big\}$$

We would like to write the diffusion operator in a way similiar to *curl-curl* form,

$$\Delta\mathbf{u} = -\nabla\times\nabla\times\mathbf{u} + \nabla(\nabla\cdot\mathbf{u}),$$

used in the INS equations. Expanding the expression for $\mathcal{D}_u$ gives

$$\begin{aligned}\mathcal{D}_u &= \partial_x(2\nu_T u_x) + \partial_y(\nu_T u_y) + \partial_z(\nu_T u_z) + \partial_y(\nu_T v_x) + \partial_z(\nu_T w_x)\\ &= 2\nu_T u_{xx} + \nu_T u_{yy} + \nu_T u_{zz} + 2\partial_x\nu_T u_x + \partial_y\nu_T(u_y + v_x) + \partial_z\nu_T(u_z + w_x) + \nu_T(v_{xy} + w_{xz})\\ &= \nu_T\Delta u + 2\partial_x\nu_T u_x + \partial_y\nu_T(u_y + v_x) + \partial_z\nu_T(u_z + w_x)\end{aligned}$$

where we have used $v_{xy} + w_{xz} = -u_{xx}$ Thus we can write

$$\mathcal{D}_u = \nu_T\Delta u - 2\partial_x\nu_T(v_y + w_z) + \partial_y\nu_T(u_y + v_x) + \partial_z\nu_T(u_z + w_x)$$

This leads in two dimensions to the *curl-curl* form

$$\mathcal{D}_u^{(2D)} = \nu_T(-v_{xy} + u_{yy}) - 2\partial_x\nu_T v_y + \partial_y\nu_T(u_y + v_x) \tag{68}$$

$$\mathcal{D}_v^{(2D)} = \nu_T(\ \ v_{xx} - u_{xy}) - 2\partial_y\nu_T u_x + \partial_x\nu_T(v_x + u_y) \tag{69}$$

while in three dimensions,

$$\mathcal{D}_u = \nu_T(-v_{xy} - w_{xz} + u_{yy} + u_{zz}) - 2\partial_x\nu_T(v_y + w_z) + \partial_y\nu_T(u_y + v_x) + \partial_z\nu_T(u_z + w_x) \tag{70}$$

$$\mathcal{D}_v = \nu_T(\ \ v_{xx} - u_{xy} - w_{yz} + v_{zz}) - 2\partial_y\nu_T(w_z + u_x) + \partial_z\nu_T(v_z + w_y) + \partial_x\nu_T(v_x + u_y) \tag{71}$$

$$\mathcal{D}_w = \nu_T(\ \ w_{xx} + w_{yy} - u_{xz} - v_{yz}) - 2\partial_z\nu_T(u_x + v_y) + \partial_x\nu_T(w_x + u_z) + \partial_y\nu_T(w_y + v_z) \tag{72}$$

These *curl-curl* forms (68-72) remove the normal derivatives of the normal components of the velocity from $\mathbf{n}\cdot(\mathcal{D}_u, \mathcal{D}_v, \mathcal{D}_w)$. For example, the expression for $\mathcal{D}_u$ contains no x-derivatives of u while $\mathcal{D}_v$ contains no y-derivatives of v.

The alternative conservative form is

$$\begin{aligned}\mathcal{D}_u &= \partial_x(-2\nu_T(v_y + w_z)) + \partial_y(\nu_T u_y) + \partial_z(\nu_T u_z) + \partial_y(\nu_T v_x) + \partial_z(\nu_T w_x)\\ \mathcal{D}_v &= \partial_x(\nu_T v_x) + \partial_y(-2\nu_T(u_x + w_z)) + \partial_z(\nu_T v_z) + \partial_x(\nu_T u_y) + \partial_z(\nu_T w_y)\\ \mathcal{D}_w &= \partial_x(\nu_T w_x) + \partial_y(\nu_T w_y) + \partial_z(-2\nu_T(u_x + v_y)) + \partial_y(\nu_T v_z) + \partial_x(\nu_T u_z)\end{aligned}$$

14 Steady state line solver

We first consider the case of a rectangular grid in two space dimensions.

The implicit line solver uses local time stepping where the local time step $\Delta_{\mathbf{i}}$ is defined from

$$\Delta t_{\mathbf{i}} = ...$$

We solve implicit scalar-tri-diagonal systems in each spatial direction. Along the x-direction we solve a tridiagonal system for U, followed by a tridiagonal system for V for the equations

$$\begin{aligned}
\frac{U_{\mathbf{i}}^{n+1}-U_{\mathbf{i}}^{n}}{\Delta t_{\mathbf{i}}} &= -\Big\{U^n D_{0x}U^{n+1} + V^n D_{0y}U^n + D_{0x}P^n\Big\} \\
&\quad + \nu\Big\{D_{+x}D_{-x}U^{n+1} + (U_{j+1}^n - 2U^{n+1} + U_{j-1}^n)/h_y^2\Big\} \\
&\quad + \nu_A(\mathbf{U}^n)\Big\{\Delta_{+x}\Delta_{-x}U^{n+1} + (U_{j+1}^n - 2U^{n+1} + U_{j-1}^n)\Big\} \\
\frac{V_{\mathbf{i}}^{n+1}-V_{\mathbf{i}}^{n}}{\Delta t_{\mathbf{i}}} &= -\Big\{U^n D_{0x}V^{n+1} + V^n D_{0y}V^n + D_{0y}P^n\Big\} \\
&\quad + \nu\Big\{D_{+x}D_{-x}V^{n+1} + (V_{j+1}^n - 2V^{n+1} + V_{j-1}^n)/h_y^2\Big\} \\
&\quad + \nu^{(2)}(\mathbf{U}^n)\Big\{\Delta_{+x}\Delta_{-x}V^{n+1} + (V_{j+1}^n - 2V^{n+1} + V_{j-1}^n)\Big\}
\end{aligned}$$

Here $\nu^{(2)}(\mathbf{U}^n)$ is the coefficient of the artificial dissipation. There is also a self-adjoint version of the artificial dissipation,

$$\begin{aligned}
\beta_{SA} &= \Delta_{+x}\Big[\nu^{(2)}_{i_1-\frac{1}{2}}\Delta_{-x}\Big]U + \Delta_{+y}\Big[\nu^{(2)}_{i_2-\frac{1}{2}}\Delta_{-y}\Big]U \\
&= \nu^{(2)}_{i_1+\frac{1}{2}}(U_{i_1+1}-U) - \nu^{(2)}_{i_1-\frac{1}{2}}(U-U_{i_1-1}) \\
&+ \nu^{(2)}_{i_2+\frac{1}{2}}(U_{i_2+1}-U) - \nu^{(2)}_{i_2-\frac{1}{2}}(U-U_{i_2-1}) \\
&= \nu^{(2)}_{i_1+\frac{1}{2}}U_{i_1+1} + \nu^{(2)}_{i_1-\frac{1}{2}}U_{i_1-1} + \nu^{(2)}_{i_2+\frac{1}{2}}U_{i_2+1} + \nu^{(2)}_{i_2-\frac{1}{2}}U_{i_2-1} \\
&- \Big(\nu^{(2)}_{i_1+\frac{1}{2}} + \nu^{(2)}_{i_1-\frac{1}{2}} + \nu^{(2)}_{i_2+\frac{1}{2}} + \nu^{(2)}_{i_2-\frac{1}{2}}\Big)U
\end{aligned}$$

After solving in the x-direction we then solve along lines in the y-direction.

On curvilinear grids the expressions are a bit more complicated. Before discretization the equations transformed to the unit-square are

$$\begin{aligned}
u_t &= -\Big\{(ur_x + vr_y)u_r + (us_x + vs_y)u_s + r_x p_r + s_x p_s\Big\} \\
&\quad + \nu\frac{1}{J}\Big\{\partial_r(J(r_x u_x + r_y u_y)) + \partial_s(J(s_x u_x + s_y u_y))\Big\} \\
u_x &= r_x u_r + s_x u_s \\
u_y &= r_y u_r + s_y u_s \\
J &= |\partial\mathbf{x}/\partial\mathbf{r}| = x_r y_s - x_s y_r
\end{aligned}$$

We solve scalar-tridiagonal-systems in the r and s directions.

14.1 Fourth-order artificial dissipation

A fourth-order artificial dissipation is

$$\nu^{(4)}(\mathbf{U}^n)\Big\{(\Delta_{+x}\Delta_{-x})^2U + (\Delta_{+y}\Delta_{-y})^2U\Big\}$$

or in a self-adjoint form

$$
\begin{aligned}
\beta_{SA}^{(4)} &= \Delta_{+x}\Delta_{-x}\Big[\nu_{\mathbf{i}}^{(4)}\Delta_{+x}\Delta_{-x}\Big]U + \Delta_{+y}\Delta_{-y}\Big[\nu_{\mathbf{i}}^{(4)}\Delta_{+y}\Delta_{-y}\Big]U \\
&= \nu_{i_1+1}^{(4)}\Delta_{+x}\Delta_{-x}U_{i_1+1} - 2\nu_{\mathbf{i}}^{(4)}\Delta_{+x}\Delta_{-x}U + \nu_{i_1-1}^{(4)}\Delta_{+x}\Delta_{-x}U_{i_1-1} \\
&+ \nu_{i_2+1}^{(4)}\Delta_{+y}\Delta_{-y}U_{i_2+1} - 2\nu_{\mathbf{i}}^{(4)}\Delta_{+y}\Delta_{-y}U + \nu_{i_2-1}^{(4)}\Delta_{+y}\Delta_{-y}U_{i_2-1} \\
&= \nu_{i_1+1}^{(4)}U_{i_1+2} - 2\big[\nu_{i_1+1}^{(4)} + \nu_{\mathbf{i}}^{(4)}\big]U_{i_1+1} + \nu_{i_1-1}^{(4)}U_{i_1-2} - 2\big[\nu_{i_1-1}^{(4)} + \nu_{\mathbf{i}}^{(4)}\big]U_{i_1-1} \\
&+ \nu_{i_2+1}^{(4)}U_{i_2+2} - 2\big[\nu_{i_2+1}^{(4)} + \nu_{\mathbf{i}}^{(4)}\big]U_{i_2+1} + \nu_{i_2-1}^{(4)}U_{i_2-2} - 2\big[\nu_{i_2-1}^{(4)} + \nu_{\mathbf{i}}^{(4)}\big]U_{i_2-1} \\
&+ \big[\nu_{i_1+1}^{(4)} + \nu_{i_2+1}^{(4)} + 8\nu_{\mathbf{i}}^{(4)} + \nu_{i_1-1}^{(4)} + \nu_{i_2-1}^{(4)}\big]U_{\mathbf{i}}
\end{aligned}
$$

15 Verification and Validation

15.1 Flat plate boundary layer

In this section we consider the computation of the flow over a flat plate. The plate is horizontal and starts at $(x, y) = (0, 0)$. The boundary layer solution is an approximation solution to the laminar flow past a flat plate. The solution is given by (derived by Prandtl's student Blasius)

$$u = U f'(\eta),$$
$$v = \frac{1}{2}\sqrt{\frac{\nu U}{x}}\Big(\eta f' - f\Big),$$

where the similiarity variable η is defined as

$$\eta = y\sqrt{\frac{U}{\nu x}},$$

and where f satisfies the 3rd order ODE:

$$f f'' + 2f''' = 0, \qquad f(0) = 0,\ f'(0) = 0,\ f'(\infty) = 1.$$

This problem can be solved as a shooting problem with initial condition

$$f''(0) \approx 0.3320573362151946$$

Note that v only makes sense if $\sqrt{\frac{\nu U}{x}}$ is small which implies ν is small and x is not too small (i.e. we cannot evaluate the solution too close to the leading edge). We thus start the computation at some offset value $x = x_0$

The thickness of the boundary layer is

$$\delta(x) \approx C_\delta\sqrt{\frac{\nu x}{U}},$$

where $C_\delta \approx 5$ for $u \approx .99U$ on the edge of the boundary layer. The thickness of the boundary layer at inflow will this be $\delta(x_0)$ and we should therefore have enough grid points to resolve this layer.

This boundary solution is evaluated in the class `BoundaryLayerProfile`. Since the solution is only approximate the errors will not go zero as the mesh is refined. The errors should become smaller, however, as $\sqrt{\frac{\nu U}{x_0}} \to 0$, e.g. if $\nu \to 0$ or $x_0 \to \infty$.

The cgins script `flatPlate.cmd` can be used to solve for the flow past a flat plate.

Figure 9 shows results for the flat plate boundary layer.

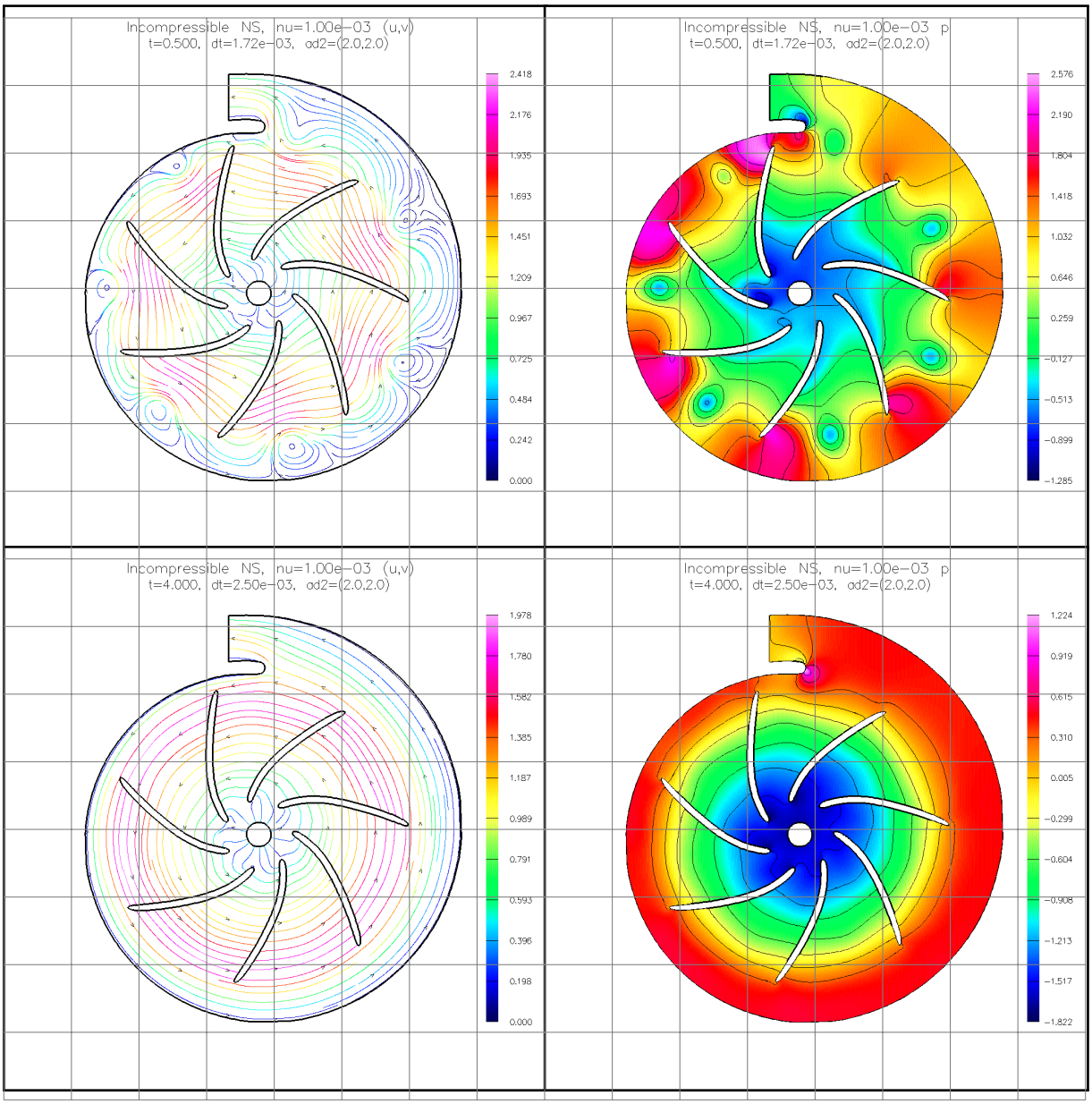

Figure 9: Flat plate boundary layer.

15.2 Heat transfer verificiation and validation

In this section we consider some verificiation and validation of the heat transfer capabilities.

15.2.1 Heated domain with no flow

We consider heat transfer in a domain with no fluid flow. The governing equation is thus the heat equation for the temperature,

$$T_t = \kappa \Delta T + f_T, \tag{73}$$

where f_T is the body force and $\kappa = k/(\rho C_p)$ is the thermal diffusivity.

Case: heated strip: One-dimensional heat transfer in a periodic strip with a constant body force heating,

$$T_t = \kappa T_{xx} + f_0, \quad x \in (0,1), \tag{74}$$
$$T(0,y,t) = 0, \quad T(1,y,t) = 0, \tag{75}$$
$$T(x,y,t) = T(x,y+1,t), \quad \text{(periodic in y)}, \tag{76}$$
$$T(x,y,0) = 0. \tag{77}$$

The steady state solution is

$$T_p(x) = \frac{f_0}{2\kappa} x(1-x), \tag{78}$$

with heat flux on the walls,

$$\mathcal{H} = kT_n = -\frac{f_0}{2}\frac{k}{\kappa} = -\frac{f_0}{2}\rho C_p, \qquad \text{(steady state heat flux at } x = 0 \text{ and } x = 1). \tag{79}$$

The solution to the time dependent problem is

$$T = T_p(x) + \sum_{n=1}^{\infty} a_n e^{-\kappa(n\pi)^2 t} \sin(n\pi x), \tag{80}$$
$$a_n = -2\int_0^1 T_p(x)\sin(n\pi x)dx. \tag{81}$$

We solve the problem on a square grid for $[0,1]\times[0,1]$, periodic in the y-direction. The grid is created with

```
ogen -noplot squareArg -periodic=np -order=2 -nx=32
```

Cgins is run with the command (scheme IM2)

```
cgins heatFlux -g=square32np.order2 -nu=.72 -tp=.5 -adcBoussinesq=0.  -thermalConductivity=.1 -gravity="0.
0.  0." -go=halt
```

where $f_0 = 1$, $k = .1$, $\kappa = 1$.

At $t = 10$ the peak temperature is .125000 (exact is .125) and the measured heat flux on the left wall is $\mathcal{H} = -4.999778677e-02$ (compared to the exact value of $\mathcal{H}$ = -5e-2). The numerical solution should converge in time to the exact steady-state solution since the steady state solution is a polynomial of degree 2.

Fourth-order:

```
ogen -noplot squareArg -periodic=np -order=4 -nx=32 -numGhost=3
```

Cgins is run with the command (scheme PC24)

```
cgins heatFlux -g=square32np.order4.ng3 -nu=.72 -ts=pc -tp=.5 -adcBoussinesq=0.  -thermalConductivity=.1
-gravity="0.  0.  0." -go=halt
```

At $t = 2.5$ the heat flux at the wall is $\mathcal{H} = -5.000000000e-02$. (Note that $e^{-2.5\pi^2} \approx 1.9e-11$ so that the time-dependent part of (80) is small).

Case: heated square: Two-dimensional heat transfer in the unit square with a body forcing,

$$f_T = f_0 \sin(m_x \pi x) \sin(m_y \pi y), \tag{82}$$
$$T(x, y, 0) = 0, \tag{83}$$

has solution

$$T(x, y, t) = \frac{f_0}{\alpha}(1 - e^{-\alpha t}) \sin(m_x \pi x) \sin(m_y \pi y), \tag{84}$$
$$\alpha = (m_x^2 + m_y^2)\pi^2 \kappa. \tag{85}$$

The average temperature in the 2D domain $[0, 1]^2$ is

$$\bar{T} = \begin{cases} \frac{f_0}{\alpha}(1 - e^{-\alpha t}) \ \frac{2}{m_x \pi} \ \frac{2}{m_y \pi}, & \text{for } m_x \text{ odd and } m_y \text{ odd,} \\ 0, & \text{otherwise} \end{cases}. \tag{86}$$

The average heat flux on $x = 0$ is

$$\mathcal{H}(x = 0) = \int_0^1 -kT_x dy = -k(1 - e^{-\alpha t}) \int_0^1 m_x \pi \cos(m_x \pi x) \sin(m_y \pi y) dy \tag{87}$$
$$= \begin{cases} -k(1 - e^{-\alpha t}) \ \frac{2m_x}{m_y}, & \text{for } m_y \text{ odd,} \\ 0, & \text{for } m_y \text{ even.} \end{cases}. \tag{88}$$

Solving this problem with Cgins on grid `square32.order2` to steady state ($\nu = .72$, $\kappa = 1$, $k = .1$) gives a peak temperature of $T_{\max} = 1.000804$ (true value $T_{\max} = 1$), the average (integrated) temperature of $\bar{T} = .4050$ (true value $\bar{T} = (2/\pi)^2 \approx .4052847$) and the average (summed not integrated – fix me) heat flux of $\mathcal{H}(x = 0) = -.1946$ (true value $\mathcal{H}(0) = -.2$).

Case: heated box: Three-dimensional heat transfer in the unit square with a body forcing,

$$f_T = f_0 \sin(m_x \pi x) \sin(m_y \pi y) \sin(m_z \pi z), \tag{89}$$
$$T(x, y, z, 0) = 0, \tag{90}$$

has solution

$$T(x, y, t) = \frac{f_0}{\alpha}(1 - e^{-\alpha t}) \sin(m_x \pi x) \sin(m_y \pi y) \sin(m_z \pi z), \tag{91}$$
$$\alpha = (m_x^2 + m_y^2 + m_z^2)\pi^2 \kappa. \tag{92}$$

The average temperature in the 3D domain $[0, 1]^3$ is

$$\bar{T} = \begin{cases} \frac{f_0}{\alpha}(1 - e^{-\alpha t}) \ \frac{2}{m_x \pi} \ \frac{2}{m_y \pi} \ \frac{2}{m_z \pi} & \text{for } m_x \text{ odd and } m_y \text{ odd and } m_z y \text{ odd,} \\ 0, & \text{otherwise} \end{cases}. \tag{93}$$

The average heat flux on the face $x = 0$ is

$$\mathcal{H}(x = 0) = \int_0^1 \int_0^1 -kT_x dy dz \tag{94}$$
$$= -k(1 - e^{-\alpha t}) \int_0^1 \int_0^1 -km_x \pi \cos(m_x \pi x) \sin(m_y \pi y) sin(m_z \pi z) dy dz \tag{95}$$
$$= \begin{cases} -k(1 - e^{-\alpha t}) \ \frac{4m_x}{\pi m_y m_z}, & \text{for } m_y \text{ odd and } m_z \text{ odd,} \\ 0, & \text{for } m_y \text{ even.} \end{cases}. \tag{96}$$

Solving this problem with Cgins on grid `box32.order2` to steady state ($\nu = .72$, $\kappa = 1$, $k = .1$) gives a peak temperature of $T_{\max} = 1.000804$ (true value $T_{\max} = 1$), the average (integrated) temperature of $\bar{T} = .2576$ (true value $\bar{T} = (2/\pi)^3 \approx .258012$) and the average (summed not integrated – fix me) heat flux of $\mathcal{H}(x = 0) = -.120$ (true value $\mathcal{H}(0) = -.127324$).

Case: steady heated box: Steady state heating of a box. If we choose a constant source of heat in the sub-region $\mathcal{R} = [.5(1-d), .5(1+d)]^3$ (i.e. a sub-cube of width d in each direction and volume d^3) of the unit box $[0,1]^3$,

$$f_T = f_0, \quad \text{for } \mathbf{x} \in \mathcal{R}, \tag{97}$$

then in steady state the total heat flux through the boundary will be

$$\int_{\partial\mathcal{B}} kT_n dS = -\frac{k}{\kappa}\int_{\mathcal{B}} f_T d\mathbf{x}, \tag{98}$$

$$= -\frac{k}{\kappa} f_0 d^3. \tag{99}$$

16 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 error $\propto h^\sigma$. The rate is estimated by a least squares fit to the data.

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

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

The 3D trigonometric solution is

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

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

Tables (2-6) show results from running Cgins on various grids.

grid	N	p	u	v	$\mathbf{u}$	$\nabla \cdot \mathbf{u}$
square20	20	2.9×10^{-1}	3.4×10^{-2}	3.4×10^{-2}	3.4×10^{-2}	5.0×10^{-1}
square30	30	9.6×10^{-2}	1.3×10^{-2}	1.2×10^{-2}	1.3×10^{-2}	1.8×10^{-1}
square40	40	4.5×10^{-2}	7.2×10^{-3}	6.7×10^{-3}	7.2×10^{-3}	8.1×10^{-2}
rate		2.69	2.23	2.34	2.24	2.61

Table 1: incompressible Navier Stokes, order=2, $\nu = 0.1$, $t = 1$, square, trig TZ, $\omega = 5.1$, $\alpha = 1$

grid	N	p	u	v	$\mathbf{u}$	$\nabla \cdot \mathbf{u}$
square16.order4	16	8.3×10^{-2}	1.2×10^{-2}	1.2×10^{-2}	1.2×10^{-2}	1.4×10^{-1}
square32.order4	32	6.5×10^{-3}	4.8×10^{-4}	3.9×10^{-4}	4.8×10^{-4}	7.6×10^{-3}
square64.order4	64	3.4×10^{-4}	2.6×10^{-5}	2.3×10^{-5}	2.6×10^{-5}	2.6×10^{-4}
rate		3.97	4.41	4.50	4.41	4.54

Table 2: incompressible Navier Stokes, order=4, $\nu = 0.1$, $t = 1$, square, trig TZ, $\omega = 5.1$, $\alpha = 1$

grid	N	p	u	v	$\nabla \cdot \mathbf{u}$
cic1	13	2.6×10^{-1}	1.5×10^{-1}	1.6×10^{-1}	5.2×10^{-1}
cic2	25	5.6×10^{-2}	2.4×10^{-2}	2.6×10^{-2}	1.3×10^{-1}
cic3	49	1.5×10^{-2}	5.1×10^{-3}	4.3×10^{-3}	2.3×10^{-2}
cic4	73	3.9×10^{-3}	1.3×10^{-3}	7.4×10^{-4}	4.9×10^{-3}
rate		2.4	2.7	3.0	2.7

Table 3: incompressible Navier Stokes, order=2, $\nu = 0.1$, $t = 1$, cic, trig TZ, $\omega = 2$

grid	N	p	u	v	$\mathbf{u}$	$\nabla \cdot \mathbf{u}$
cicb.order4	61	1.2×10^{-4}	3.8×10^{-5}	3.7×10^{-5}	3.8×10^{-5}	1.9×10^{-4}
cic.order4	121	9.6×10^{-6}	1.7×10^{-6}	2.1×10^{-6}	2.1×10^{-6}	9.6×10^{-6}
cic2.order4	241	6.2×10^{-7}	1.0×10^{-7}	1.2×10^{-7}	1.2×10^{-7}	1.0×10^{-6}
rate		3.86	4.30	4.18	4.19	3.78

Table 4: incompressible Navier Stokes, order=4, $\nu = 0.1$, $t = 1$, cic, trig TZ, $\omega = 1$, $\alpha = 1$

grid	N	p	u	v	w	$\nabla \cdot \mathbf{u}$
box10	10	2.0×10^{-2}	2.0×10^{-3}	2.1×10^{-3}	2.1×10^{-3}	1.7×10^{-2}
box20	20	5.0×10^{-3}	3.4×10^{-4}	3.1×10^{-4}	3.1×10^{-4}	2.7×10^{-3}
box30	30	2.4×10^{-3}	1.3×10^{-4}	1.0×10^{-4}	1.0×10^{-4}	8.1×10^{-4}
rate		1.9	2.5	2.8	2.8	2.8

Table 5: incompressible Navier Stokes, order=2, $\nu = 0.1$, $t = 1$, box, trig TZ, $\omega = 2$

grid	N	p	u	v	w	$\nabla \cdot \mathbf{u}$
box8.order4	8	1.4×10^{-3}	2.9×10^{-4}	2.6×10^{-4}	2.6×10^{-4}	1.2×10^{-3}
box16.order4	16	3.8×10^{-5}	8.4×10^{-6}	7.8×10^{-6}	7.8×10^{-6}	6.6×10^{-5}
box32.order4	32	3.6×10^{-6}	1.7×10^{-7}	1.8×10^{-7}	1.8×10^{-7}	1.6×10^{-6}
rate		4.3	5.4	5.2	5.2	4.8

Table 6: incompressible Navier Stokes, order=4, $\nu = 0.1$, $t = 1$, box, trig TZ, $\omega = 2$

grid	N	p	u	v	w	$\nabla \cdot \mathbf{u}$
sib1	17	2.8×10^{-1}	2.1×10^{-1}	1.4×10^{-1}	1.2×10^{-1}	6.2×10^{-1}
sib2	33	6.2×10^{-2}	5.0×10^{-2}	3.0×10^{-2}	1.9×10^{-2}	1.9×10^{-1}
sib2a	49	3.0×10^{-2}	1.7×10^{-2}	8.8×10^{-3}	1.2×10^{-2}	7.8×10^{-2}
rate		2.1	2.4	2.6	2.2	1.9

Table 7: incompressible Navier Stokes, order=2, $\nu = 0.1$, $t = 1$, sib, trig TZ, $\omega = 2$

grid	N	p	u	v	w	$\mathbf{u}$	$\nabla \cdot \mathbf{u}$
sib1.order4	30	1.0×10^{-2}	1.6×10^{-2}	8.8×10^{-3}	7.5×10^{-3}	1.6×10^{-2}	8.3×10^{-2}
sib1a.order4	60	8.2×10^{-4}	9.9×10^{-4}	4.5×10^{-4}	6.6×10^{-4}	9.9×10^{-4}	6.2×10^{-3}
sib2.order4	80	1.2×10^{-4}	8.7×10^{-5}	5.0×10^{-5}	7.8×10^{-5}	8.7×10^{-5}	7.8×10^{-4}
rate		4.38	5.08	5.10	4.44	5.08	4.57

Table 8: incompressible Navier Stokes, order=4, $\nu = 0.05$, $t = 1$, sib, trig TZ, $\omega = 1$, $\alpha = 1$

grid	N	p	u	v	w	$\mathbf{u}$	$\nabla \cdot \mathbf{u}$
curvedPipe1	1	1.4×10^{-3}	7.0×10^{-4}	1.2×10^{-3}	4.9×10^{-4}	1.2×10^{-3}	4.4×10^{-2}
curvedPipe2	2	4.6×10^{-4}	2.0×10^{-4}	2.3×10^{-4}	1.4×10^{-4}	2.3×10^{-4}	1.5×10^{-2}
curvedPipe3	3	2.6×10^{-4}	9.4×10^{-5}	8.8×10^{-5}	1.1×10^{-4}	1.1×10^{-4}	8.9×10^{-3}
rate		1.51	1.82	2.37	1.39	2.16	1.46

Table 9: INS, curvedPipe, order=2, $\nu = 0.025$, $t = 0.1$, cdv=1, poly TZ

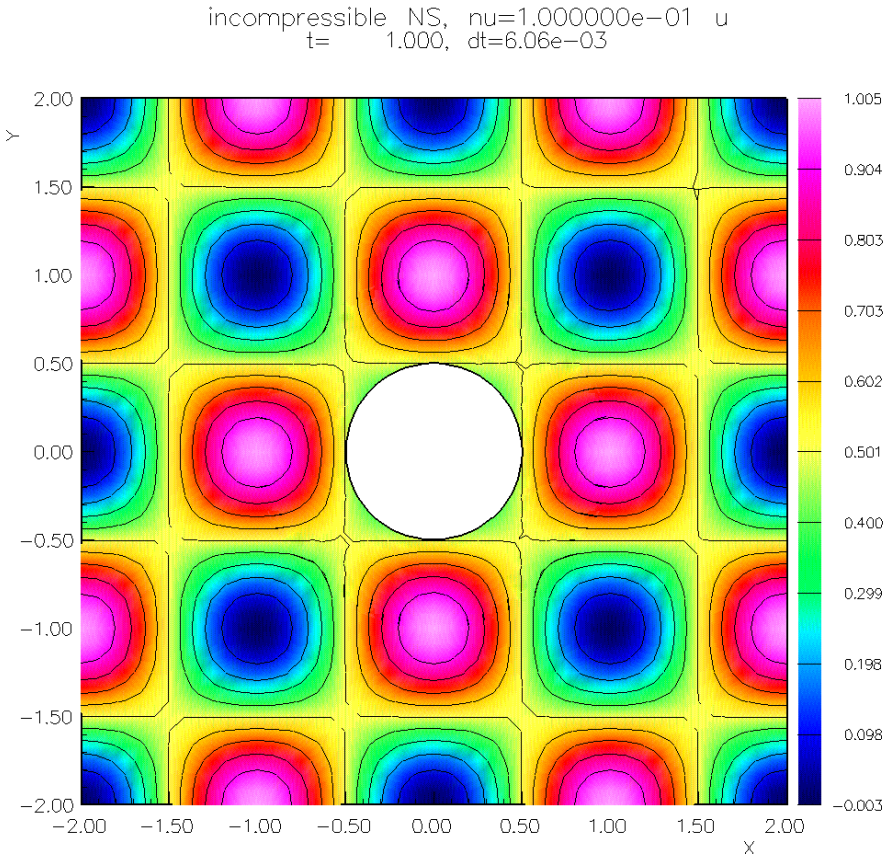

Figure 10: Incompressible N-S, twilight zone solution for convergence test

17 Performance Results

The normalized measure of CPU, *time-to-solution* (TTS), defined below, can be used to compare results from different time-stepping schemes, different numbers of processors and even different grids. Smaller values of TTS correspond to faster schemes.

$$
\begin{aligned}
\mathcal{N}_g &= \text{number of grid points,} \\
N_p &= \text{number of processors,} \\
N_s &= \text{number of time steps,} \\
\overline{\Delta t} &= t_{\text{final}}/N_s = \text{average time-step,} \\
\Delta t_a &= \text{velocity-scale/average grid spacing} = \text{time-step scale for advection,} \\
\text{TPS} &= \text{total-wall-clock-time}/N_s = \text{average CPU time (s) per step,} \\
\text{TPSM} &= \text{average CPU time (s) per step per million grid points per processor,} \\
&= N_p \times \text{TPS}/(\mathcal{N}_g/10^6), \\
\text{TTS} &= \text{normalized average CPU time (s) to solve,} \\
&= \text{TPSM} \times \Delta t_a/\overline{\Delta t}
\end{aligned}
$$

The *normalized memory usage*, RPG, measures the number of *reals-per-grid-point* (i.e. the equivalent number of double precision values (8 bytes, 64 bits) per grid point) for the entire memory use.

$$\text{RPG} = (\text{total memory usage (btyes)})/(\text{number-of-grid-points} \times (8 \text{ bytes/real}))$$

17.1 Performance: two cylinders in a channel

Grid	Pts	Method	P-solver	N_p	TPSM	TTS	RPG
tcilce16	1.4M	IM22	BICGS(3)	serial	18	54	200
		PC22	BICGS(3)	serial	9.6	62	67
		IM24	BICGS(3)	serial	32	170	314
		IM24	MG	serial	5.5	29	171
		IM24MG	MG	serial	3.9	20	49
		AFS22	MG	serial	11	16	27
		AFS24	MG	serial	13	21	31

Table 10: Performance of Cgins for different time-stepping methods and different linear solvers for the pressure equation. TTS is the normalized time-to-solution. RPG is the total memory usage in reals-per-grid-point.

Grid	Pts	Method	P-solver	N_p	TPSM	TTS	RPG
tcilce64	22.M	PC24	BICGS(5)	16	840	16000	205
		PC24	MG	16	15	210	67
		IM24MG	MG	16	34	180	46
		AFS24	MG	16	34	54	32

Table 11: Performance of Cgins for different time-stepping methods and different linear solvers for the pressure equation. TTS is the normalized time-to-solution. RPG is the total memory usage in reals-per-grid-point.

Table 11 gives performance results for grid `tcilce64.order4.ml4`. NOTE: Case IM24MG-MG was run at $\nu = 10^{-4}$ while the other cases were run at $\nu = 10^{-5}$ so the value of TTS is not exactly comparable.

17.2 Performance: pitching-plunging airfoil

We consider flow past a two-dimensional Joukowsky airfoil that undergoes a pitching and plunging motion. Let $\mathcal{G}_j^{(j)}$ denote the grid for this problem with target grid spacing of $\Delta s = .05/j$.

Grid $\mathcal{G}_j^{(16)}$ has approximately 1.7M grid points. Grid $\mathcal{G}_j^{(32)}$ has approximately 6.8M grid points.

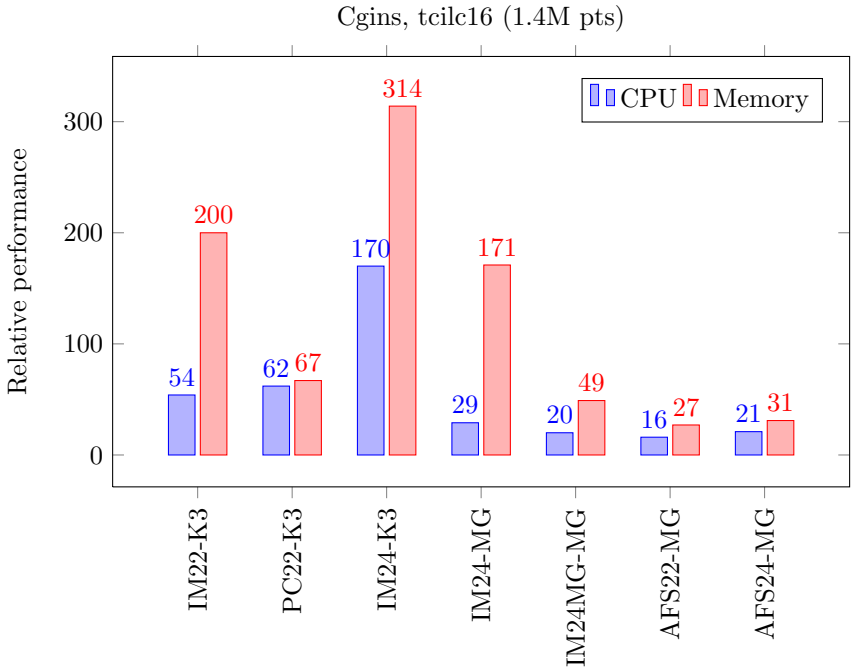

Figure 11: Performance of Cgins for different time stepping methods for flow past two cylinders in a channel. The CPU is a normalized time-to-solution (TTS). The memory usage is measured in reals-per-grid-point. K3 stands for the Krylov solver BiCGStab-ILU(3).

Grid	Pts	Method	P-solver	N_p	TPSM	TTS	RPG
joukowsky16	1.7M	AFS24	BICGS(5)	8	340	1,700	246
		AFS24	MG	8	29	140	79
joukowsky32	6.8M	AFS24	BICGS(5)	16	660	3,700	235
		AFS24	MG	16	40	230	65
		AFS24	MG	8	32	99	54

Table 12: Performance of Cgins for different time-stepping methods and different linear solvers for the pressure equation. TTS is the normalized time-to-solution. RPG is the total memory usage in reals-per-grid-point.

Grid	Pts	Method	P-solver	N_p	TPSM	TTS	RPG
joukowsky32	6.8M	AFS24	BICGS(5)	16	1100	1800	235
		AFS24	MG	16	52	76	66

Table 13: RESTART. Performance of Cgins for different time-stepping methods and different linear solvers for the pressure equation. TTS is the normalized time-to-solution. RPG is the total memory usage in reals-per-grid-point.

18 Some sample simulations

Here is a collection of interesting examples computed with the Cgins incompressible solver.

The examples include

1. Falling cylinders in an incompressible flow, Section 18.1.
2. Flow past two cylinders, Section 18.2.
3. A pitching and plunging airfoil, Section 18.3.
4. A fluttering plate, Section 18.4.
5. Centrifugal pump, Section 18.5.
6. Flow in a 2d heated room, Section 18.6.
7. Flow in a 3d heated room, Section 18.7.
8. Simulation of flow past a blood clot filter, Section 18.8.
9. Incompressible flow past a truck, Section 18.10.
10. Incompressible flow past a city scape, Section 18.11.
11. Flow past an airfoil at different Reynolds numbers, Section 18.12.
12. Flow past a pitching-plunging airfoil at different Reynolds numbers, Section 18.13.
13. Flow past two spheres, Section 18.14.
14. Flow past a deforming sphere, Section 18.15.
15. Flow past a moving 3D cylinder at different Reynolds numbers, Section 18.16.
16. Flow past a moving sphere, Section 18.17.
17. Flow past a cube in a channel, Section 18.18.
18. Flow past a rotating flattened torus, Section 18.19.
19. Flow past a rotating disk, Section 18.20.
20. Flow past a rounded blade, Section 18.21.
21. Wind turbine and tower, Section 18.22.

18.1 Falling cylinders in an incompressible flow

To run this example see `cg/ins/runs/drops/Readme`.

Figure (12) shows multiple rigid bodies failing under the influence of gravity in an incompressible flow. There is an upward flow that nearly matches the speed of the falling drops. This solution was computed using the multigrid solver for the pressure equation.

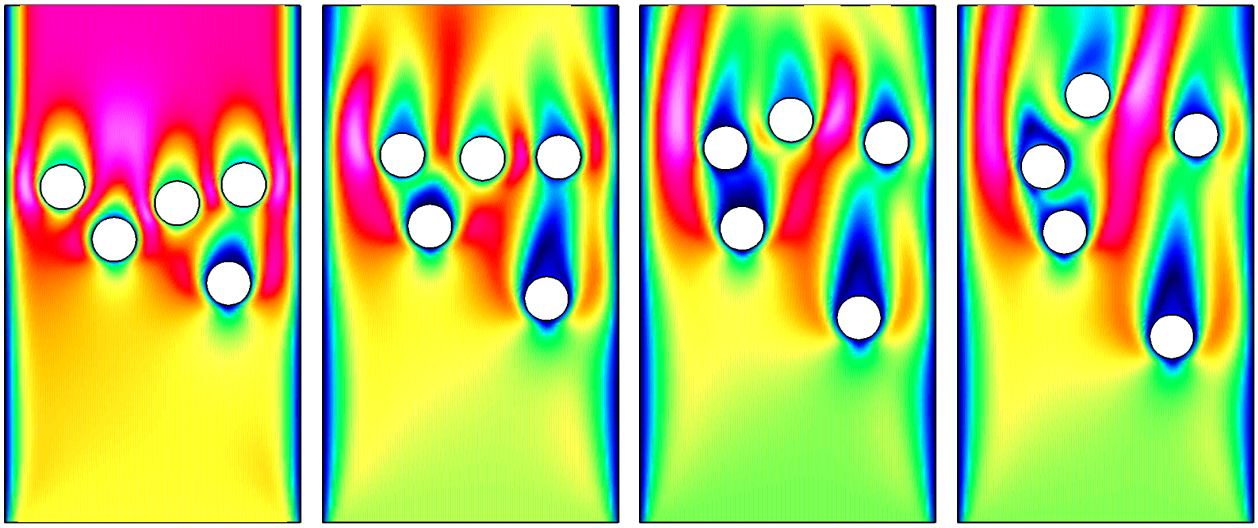

Figure 12: Multiple cylinders falling in an incompressible flow, contour plots of the flow speed. Solution at times $t = 1$, 3, 5 and 7.

18.2 Flow past two cylinders

Figure 13 shows the simulation of a high Reynolds number flow past two cylinders. The simulation used the command file `cg/ins/cmd/tcilc.cmd` and the grid was made with the ogen script `Overture/sampleGrids/tcilc.cmd`. The radius of the cylinders was 1/2. The grid had about 22 million grid points.

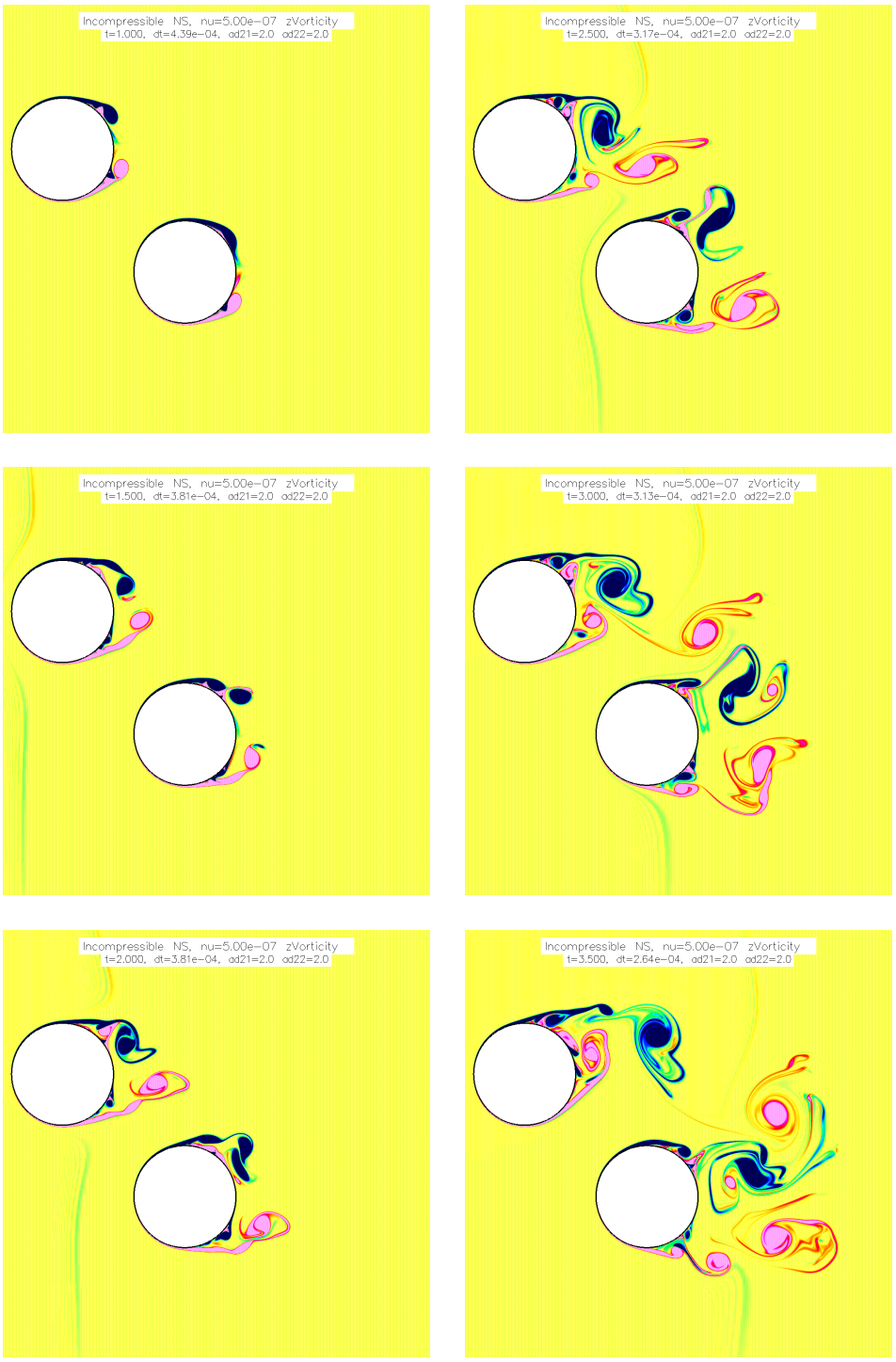

Figure 13: Flow past two cylinders in a channel computed with Cgins. Contour plots of the vorticity.

18.3 Pitching and plunging airfoil

This example shows the simulation of a pitching and plunging airfoil, The simulation used the command file `cg/ins/cmd/wing2d.cmd` and the grid was made with the ogen script `Overture/sampleGrids/joukowsky2d.cmd`. The computation was performed in parallel and demonstrates the parallel moving grid capabilities (starting with v24). The grid had $1.7M$ points.

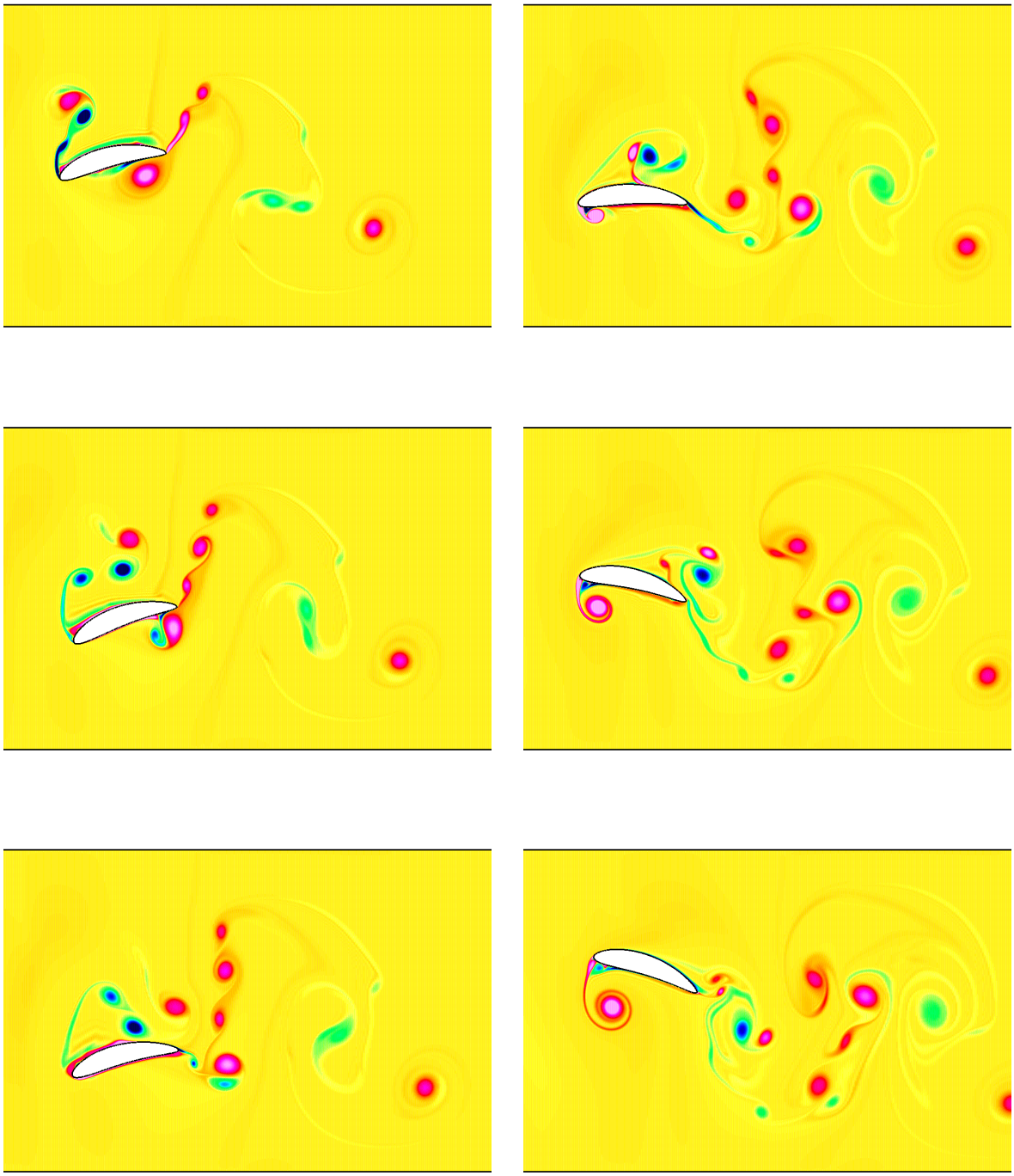

Figure 14: A pitching and plunging airfoil, computed in parallel with Cgins. Contour plots of the vorticity.

18.4 Fluttering plate

This example shows the simulation of a light plate falling under the influence of gravity. There is an upward flow with speed approximately equal to the rate at which the plate falls. The plate starts to sway from side to side and eventually also tumbles. This example demonstrates the new time stepping scheme that has been developed to treat the motion of "light" rigid bodies (v24).

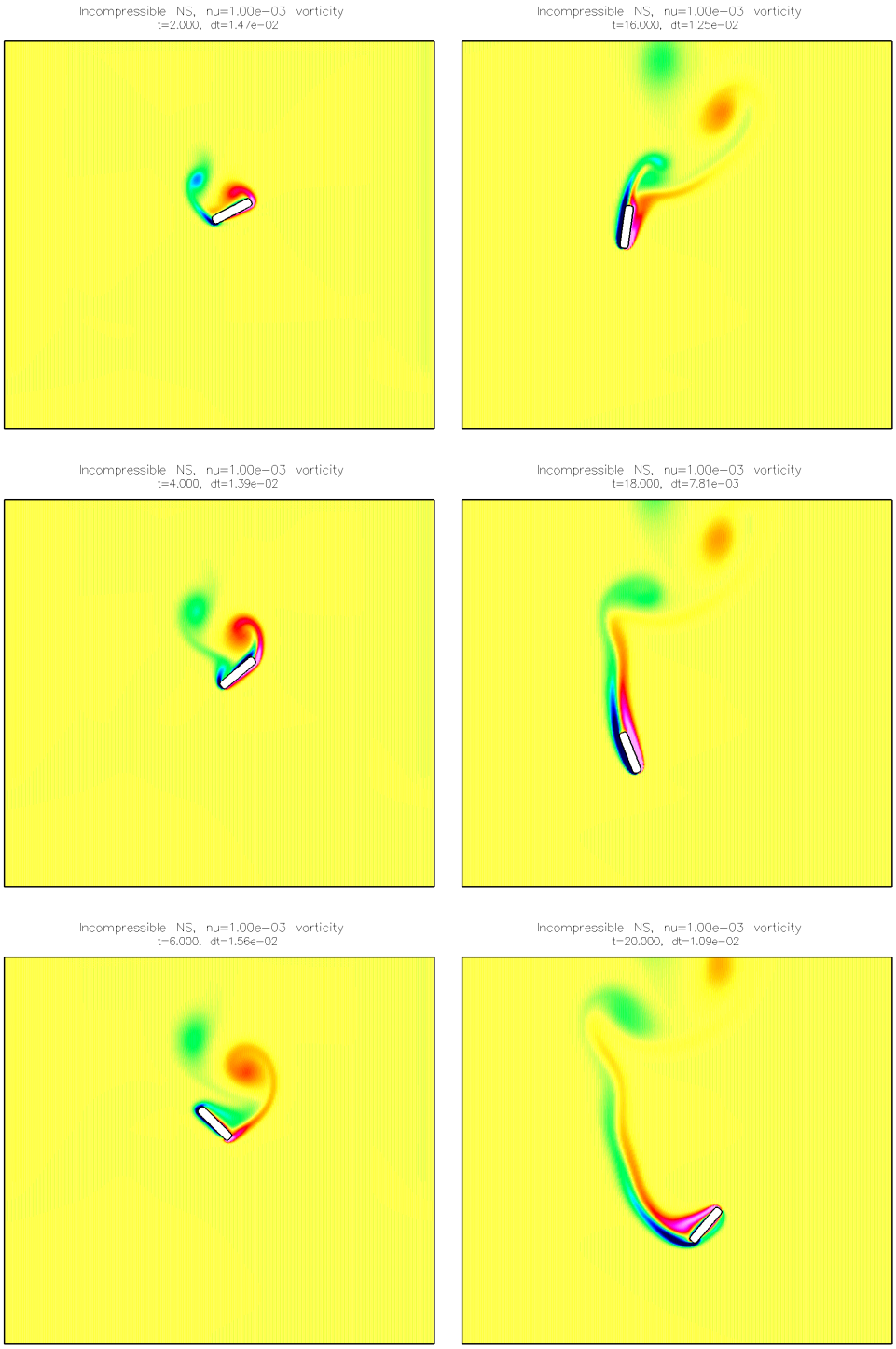

Figure 15: A fluttering plate computed with Cgins. A light plate falling due to the force of gravity in an incompressible flow. Contour plots of the vorticity.

18.5 Centrifugal pump

This example shows the simulation of a centrifugal pump. Fluid flows into the domain through the central core and is accelerated by the counter-clockwise rotating blades before exiting at the top outlet. The grid was constructed with the ogen command file `pump2dGrid.cmd`. The command file for cgins was `pump2d.cmd`. Thanks to Franck Monmont for help with the problem.

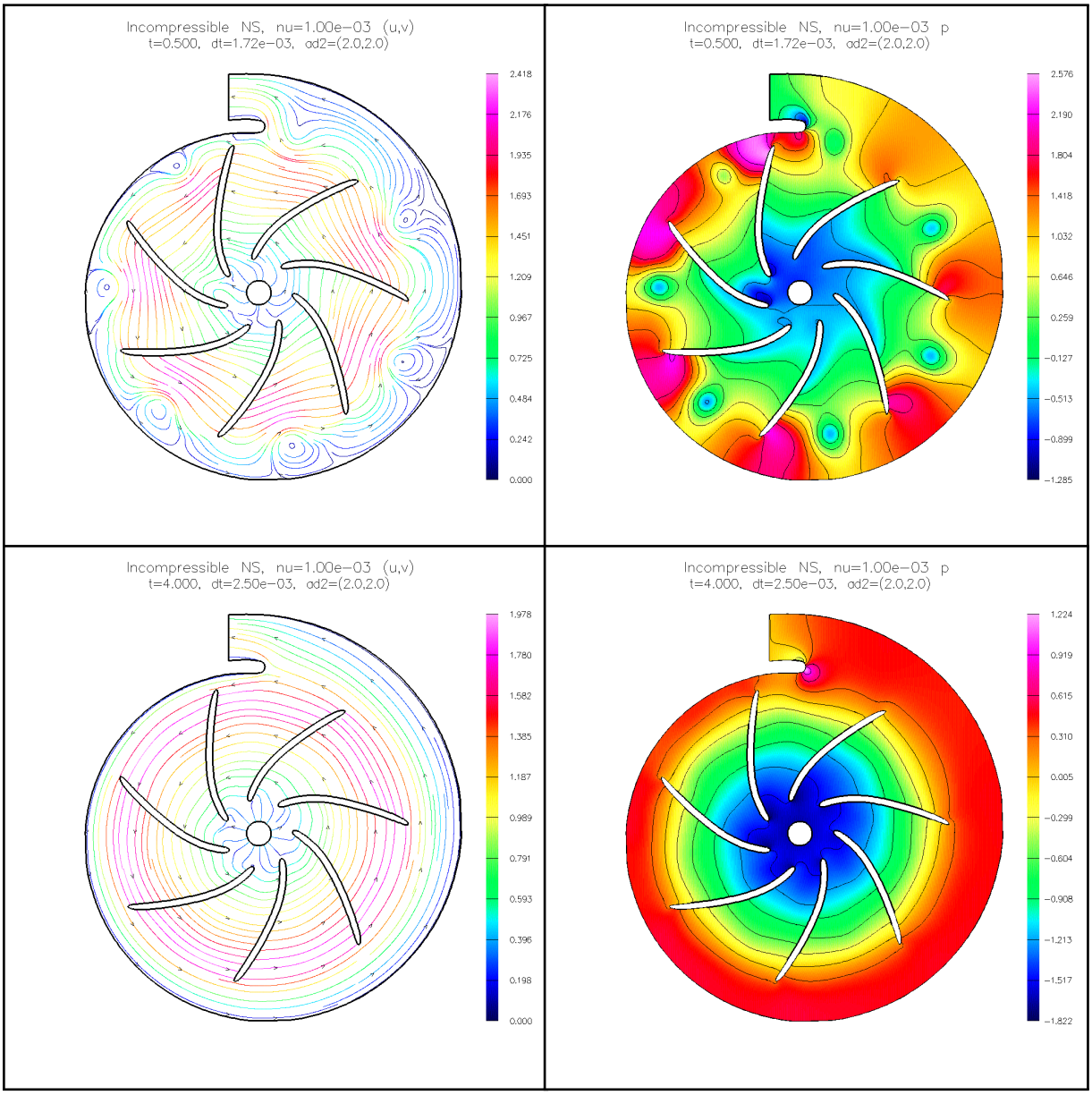

Figure 16: Centrifugal pump. Streamlines and pressure at $t = 0.5$ (top) and $t = 4.0$ (bottom).

18.6 Flow in a heated room

This example shows the simulation cool air entering a two-dimensional model of a room with a desk, computer and partition. Flow enters the domain through an inlet on the ceiling and leaves through an outlet on the ceiling. The computer under the desk is heated. We solve the INS equations with Boussinesq approximation (i.e. with gravity and buoyancy effects). The grid was constructed with the ogen command file `room2d.cmd`. The command file for cgins was `heatedRoom.cmd`.

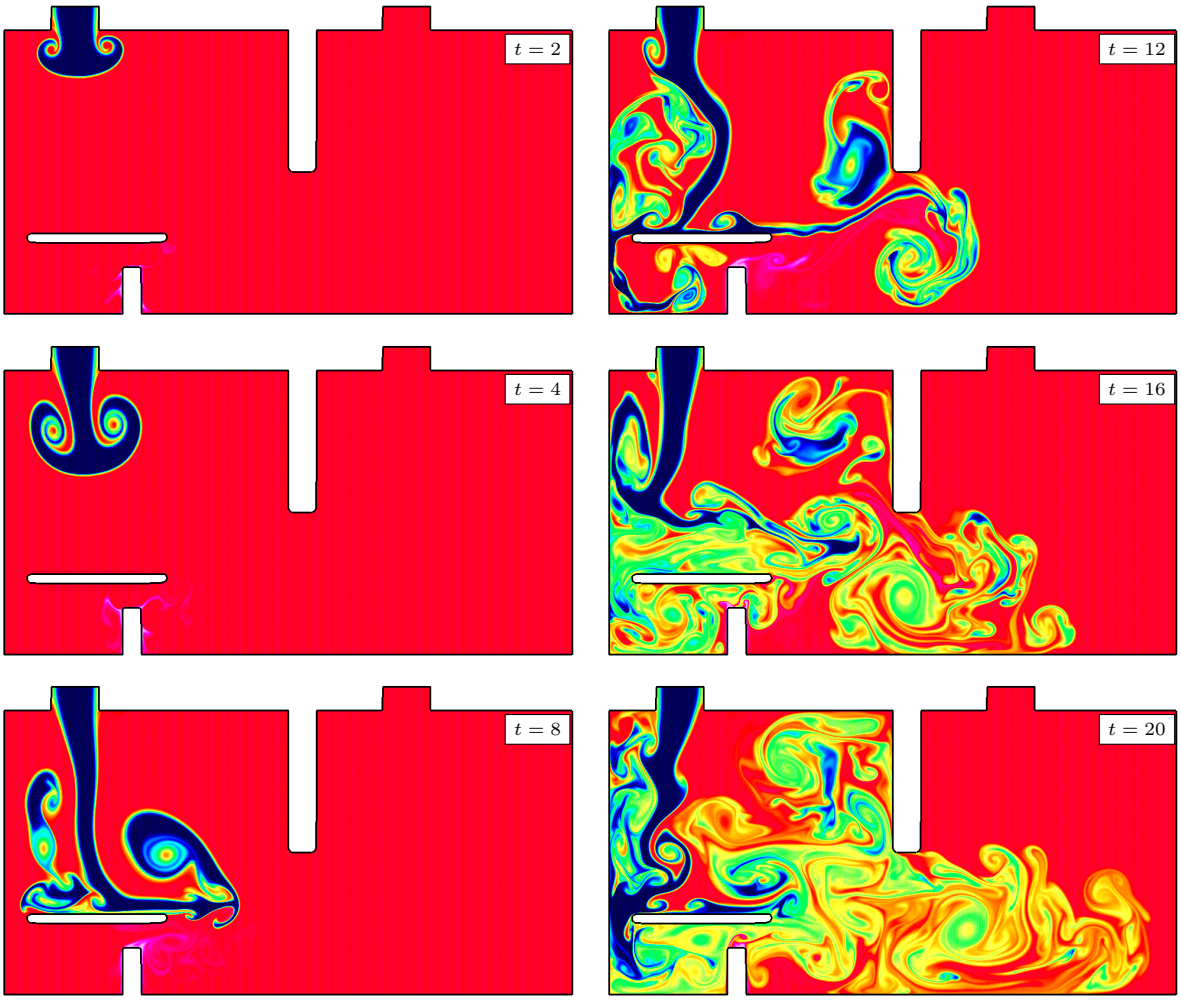

Figure 17: Results from a high-resolution simulation of cool air entering a 2D model of a room. Contours of the temperature are shown.

18.7 Flow in a 3D room

Figure 18 shows some results from a simulation of the flow in a small three-dimensional room with a desk, hot computer, inlet and outlet. Cold air enters through an inlet vent on the ceiling into a room with an initially uniform temperature. We solve the INS equations with Boussinesq approximation (i.e. with gravity and buoyancy effects). The grid was constructed with the ogen command file `room3d.cmd`. The command file for cgins was `heatedRoom3d.cmd`.

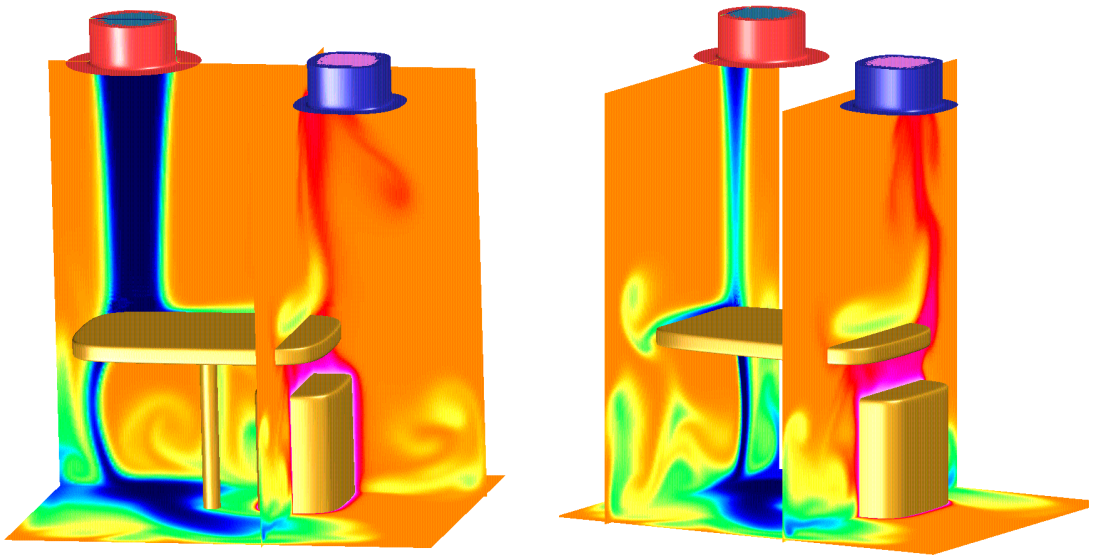

Figure 18: Results from a computation of flow in a 3D room. The figure shows contours of the temperature on selected cutting planes at $t = 15.5$ from a time dependent simulation.

18.8 Simulation of flow past a blood clot filter

Figure 19 shows results from computations of the incompressible flow in a three-dimensional cylindrical pipe ("vein") with an embedded wire frame "filter" and embedded blood clots. The computations were performed with the pseudo steady-state solution algorithm.

Further details on these simulations can be found in M.A. Singer, WDH, S.L. Wang, *Computational Modeling of Blood Flow in the Trapease Inferior Vena Cava Filter*, Journal of Vascular and Interventional Radiology, **20**, 2009.

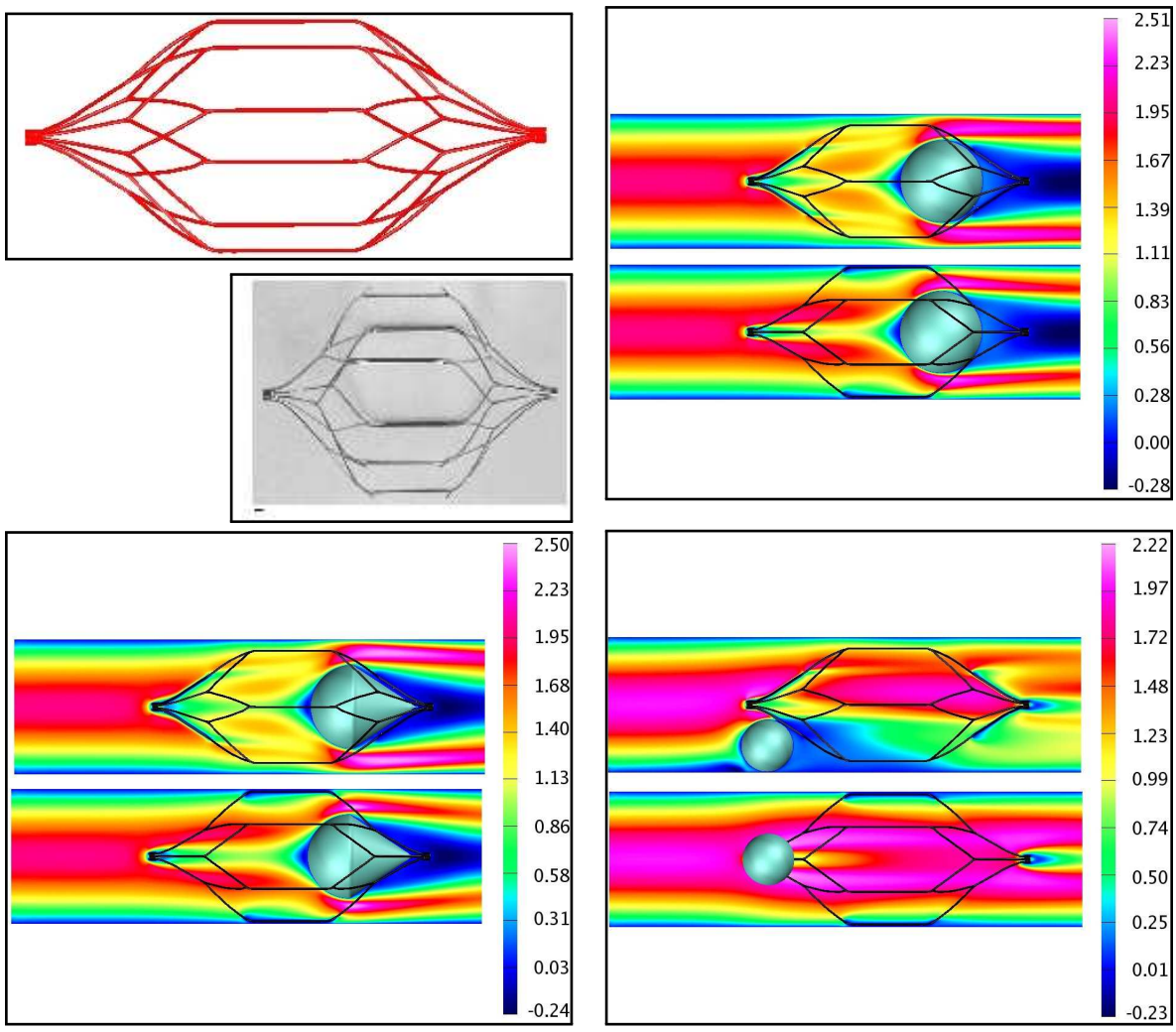

Figure 19: Flow past a blood-clot filter using cgins.

18.9 Flow in periodic channel

The flow through a periodic channel is simulated with the AFS24 scheme. Scripts for this case can be found in `cg/ins/runs/periodicChannel`. The flow is driven by a pressure gradient. An initial perturbation is added to excite the turbulence. The computations on finer grids use coarser grid solutions as initial conditions. The channel covers the domain $[-1,1] \times [0,2\pi]$. The grid $G^{(j)}$ for the channel has $64j$ grid points in the axial direction. The gridlines are stretched near the upper and lower walls by a factor of 5. The viscosity is $\nu = 10^{-5}$. For grid $G^{(16)}$ the finest grid-spacing normal to wall is apprximately 1.22e-3.

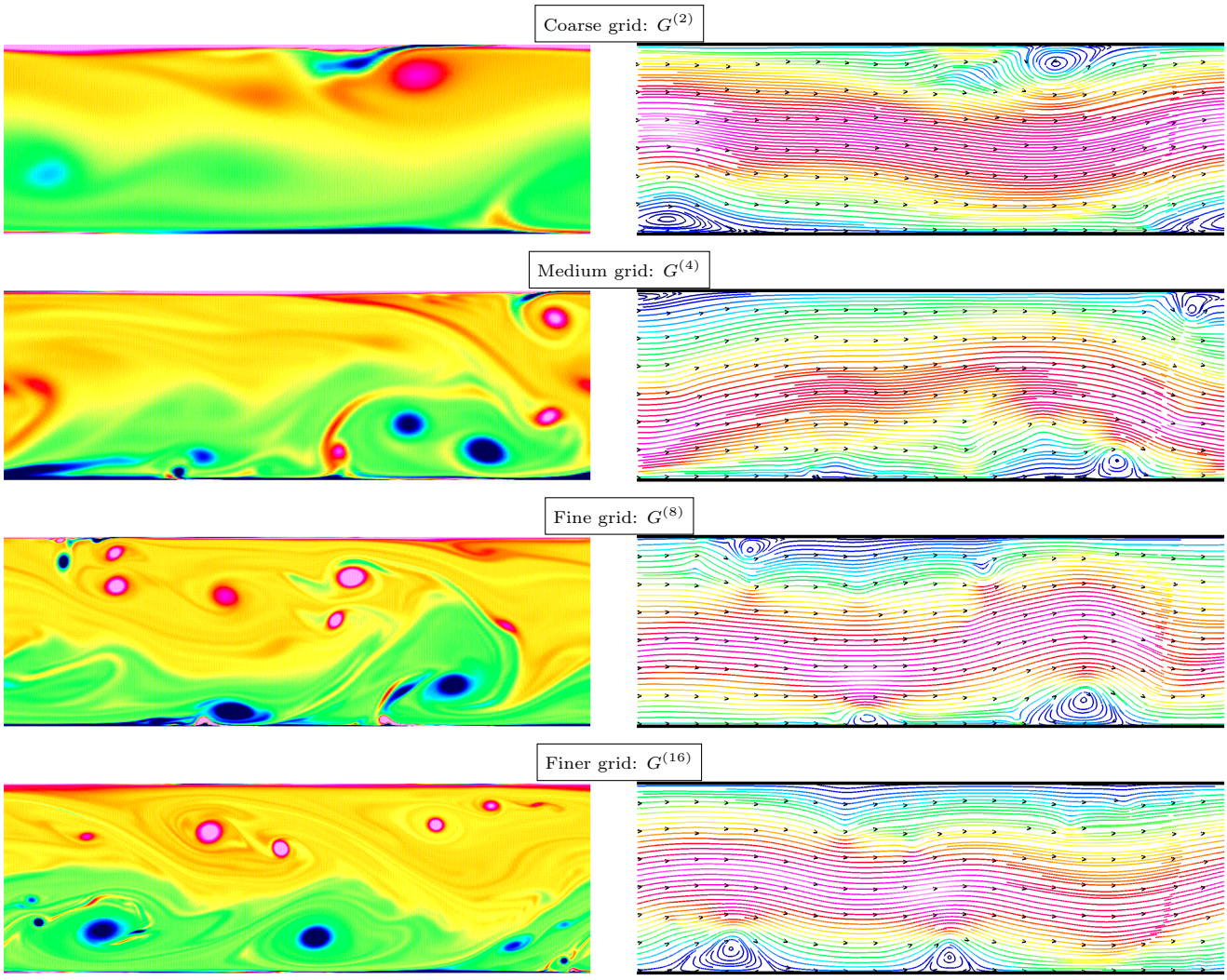

Figure 20: Turbulent flow through a periodic channel on grids of increasing resolution. Left column: vorticity. Right column: streamlines. The vorticity is shown on the scale $[-5,5]$.

Commands to generate grids:

```
ogen -noplot channelArg -order=4 -factor=2 -ml=2
ogen -noplot channelArg -order=4 -factor=4 -ml=3
ogen -noplot channelArg -order=4 -factor=8 -ml=4
ogen -noplot channelArg -order=4 -factor=16 -ml=5
ogen -noplot channelArg -order=4 -factor=32 -ml=5
ogen -noplot channelArg -order=4 -factor=64 -ml=6
```

18.10 Incompressible flow past a truck

Figure (21) shows a computation of the incompressible Navier-Stokes equations for flow past the cab of a truck. The steady state line solver was used for this computation.

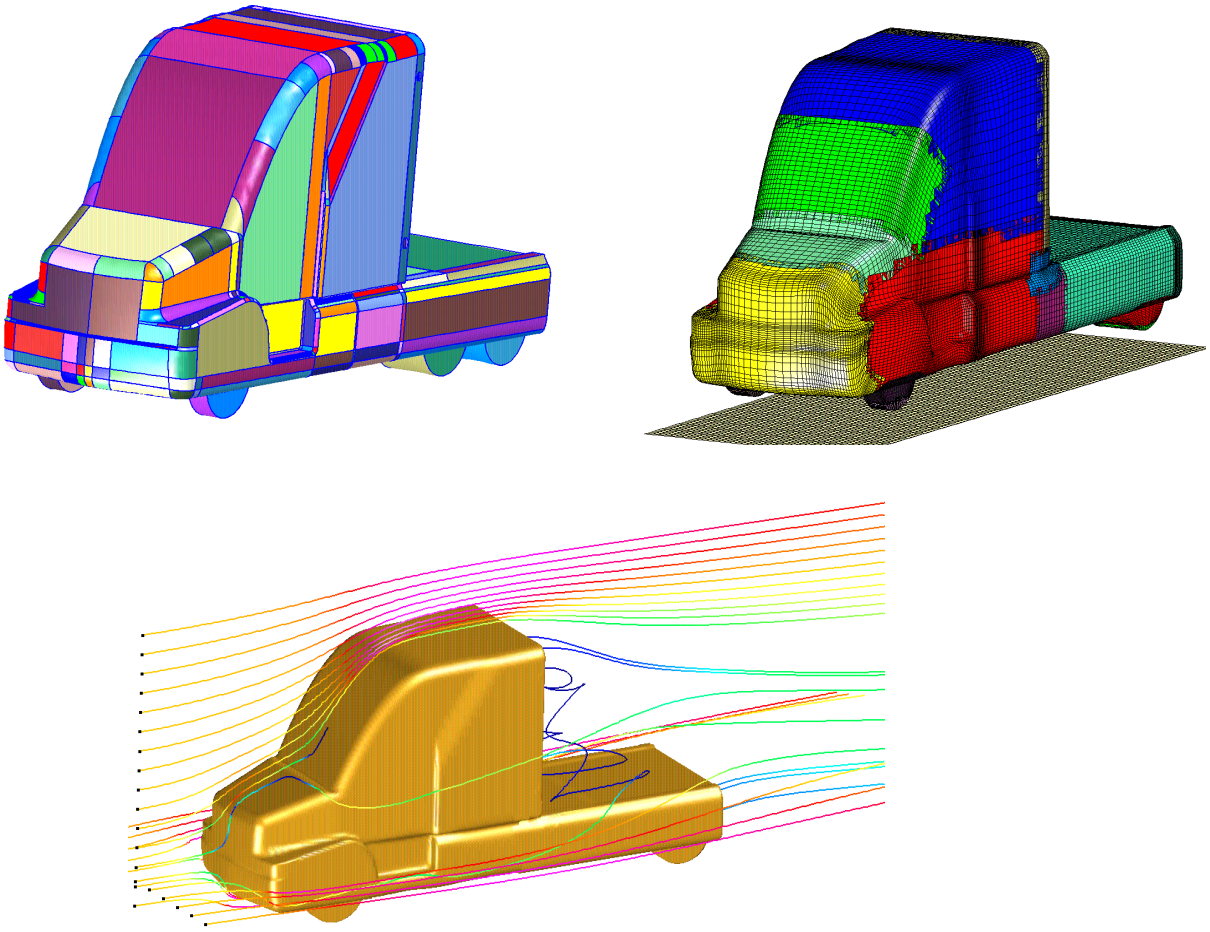

Figure 21: Incompressible flow past the cab of a truck. Shown are the CAD geometry, the grids and some tracer particles.

18.11 Incompressible flow past a city scape

Figure (22) shows a computation of the incompressible Navier-Stokes equations for flow past a city scape. The steady state line solver was used for this computation. The command file for generating this grid is `Overture/sampleGrids/multiBuildings.cmd` and the Cgins command file is `ins/cmd/multiBuildings.cmd`.

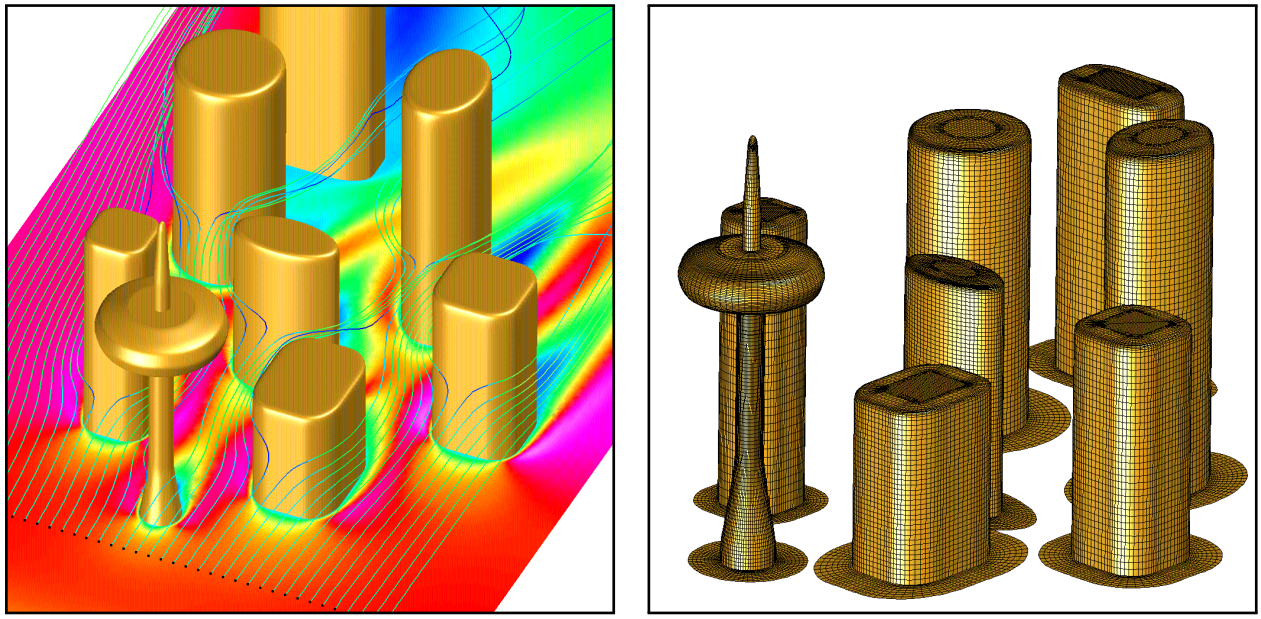

Figure 22: Incompressible flow through a city scape

Notes:

- pseudo steady-state line implicit solver, 4th-order dissipation,
- local time-stepping (spatially varying dt)
- requires 1.4GB of memory,
- cpu = 29s/step,
- 2.2 GHz Xeon, 2 GB of memory

18.12 Flow past an airfoil at different Reynolds numbers

We simulate the flow past an airfoil using the second- and fourth-order accurate versions of the AFS scheme. The second-order and fourth-order SSLES turbulence models are used. The results indicate how the flow character changes as the effective Reynolds number increases.

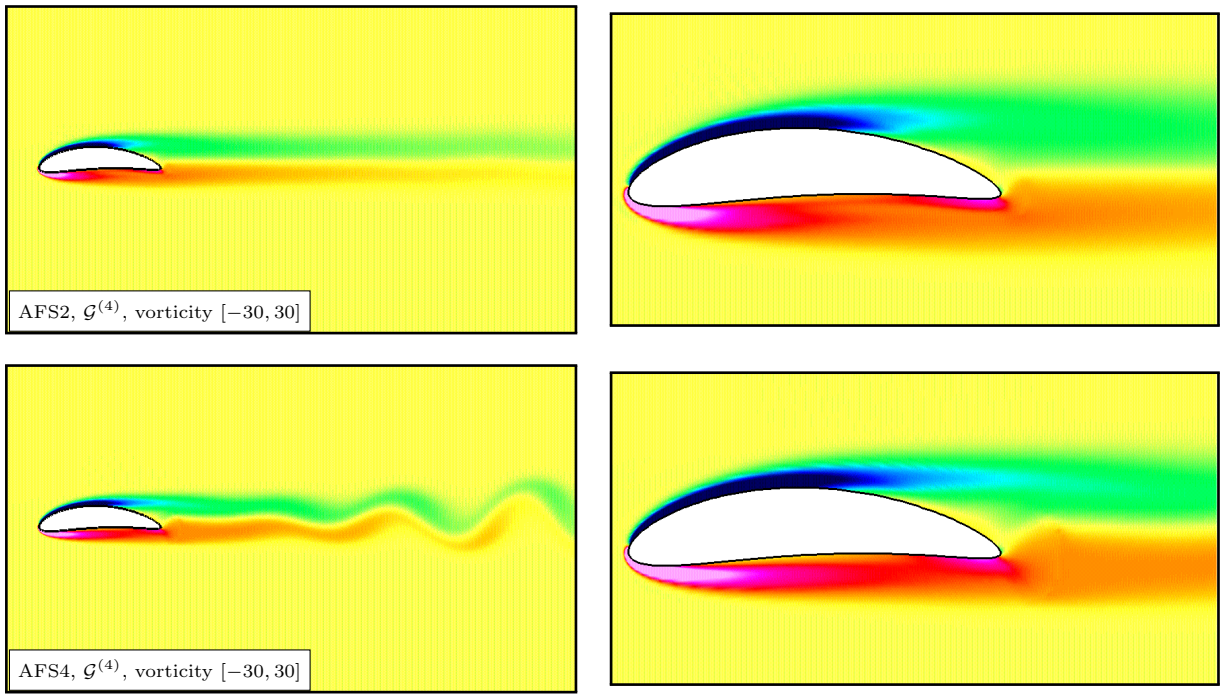

Figure 23: Flow past an airfoil on grid $\mathcal{G}^{(4)}$. Contour plots of the vorticity at $t = 10$.

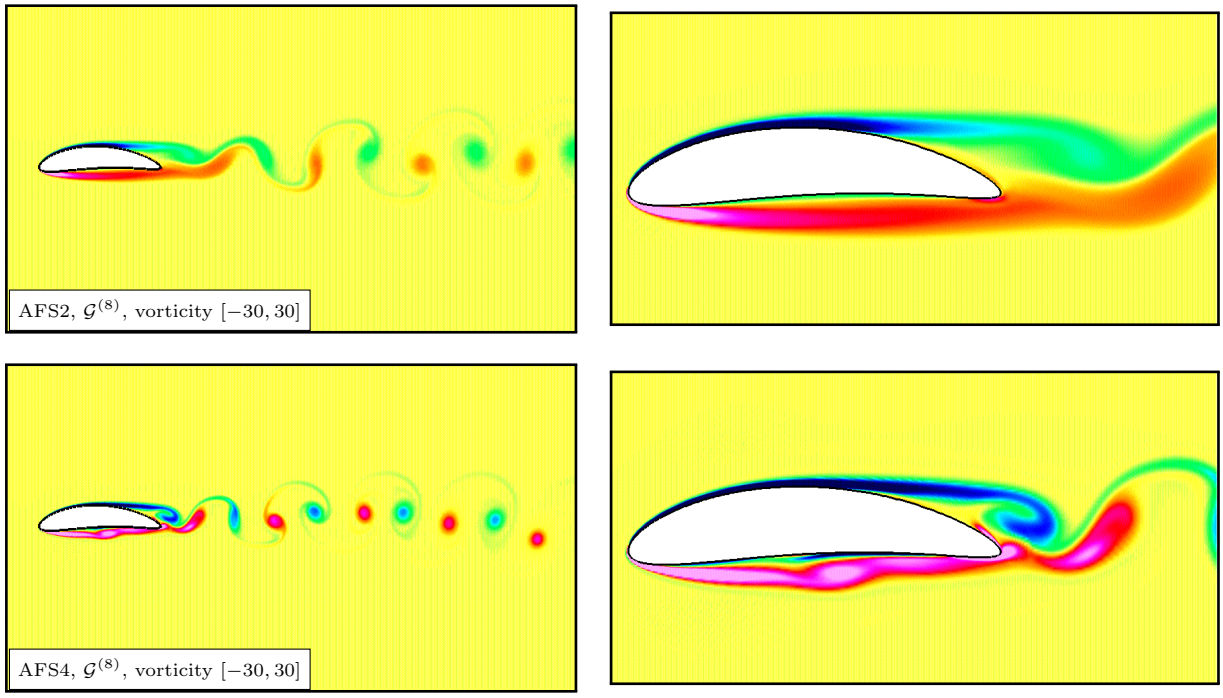

Figure 24: Flow past an airfoil on grid $\mathcal{G}^{(8)}$. Contour plots of the vorticity at $t = 10$.

The simulations used the command file `cg/ins/cmd/wing2d.cmd` and the grid was made with the ogen script `Overture/sampleGrids/joukowsky2d.cmd`. AFS2 denotes the second-order accurate AFS scheme and AFS4 is the fourth-order accurate. Grid $\mathcal{G}^{(j)}$ has a background grid spacing of about $1/(20j)$. The chord length of th airfoil is 1.

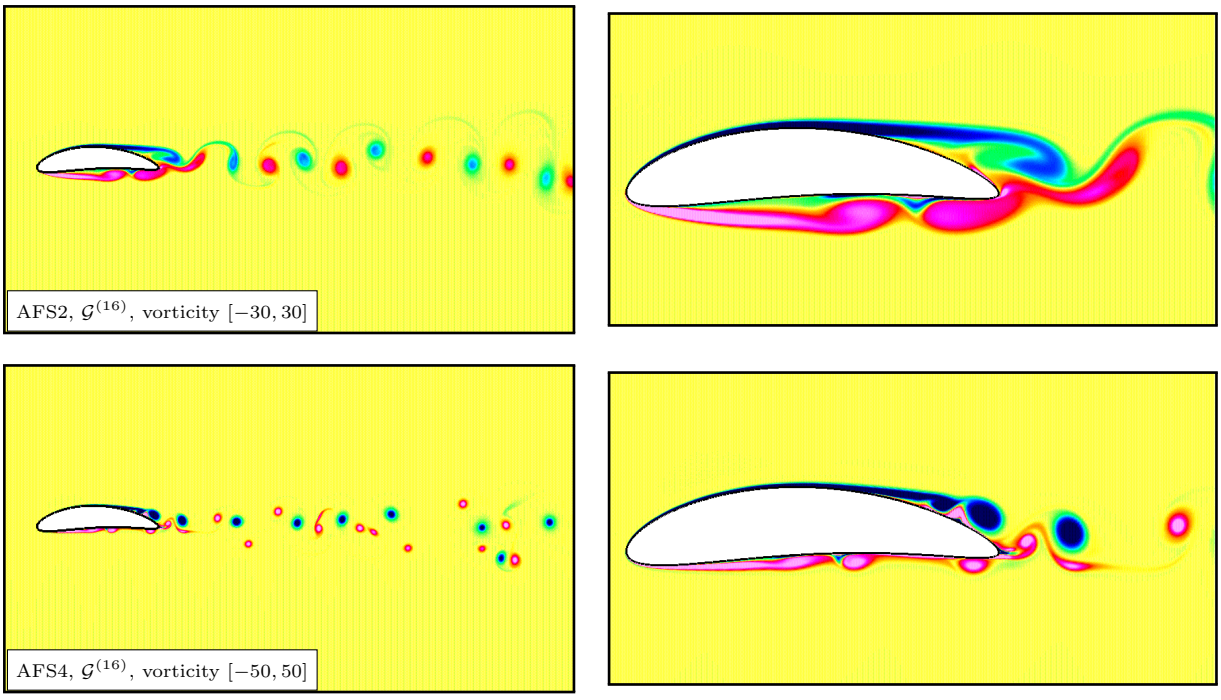

Figure 25: Flow past an airfoil on grid $\mathcal{G}^{(16)}$. Contour plots of the vorticity at $t = 10$.

18.13 Flow past a pitching-plunging airfoil at different Reynolds numbers

We simulate the flow past a pitching-plunging airfoil using the second- and fourth-order accurate versions of the AFS scheme. The second-order and fourth-order SSLES turbulence models are used. The results indicate how the flow character changes as the effective Reynolds number increases.

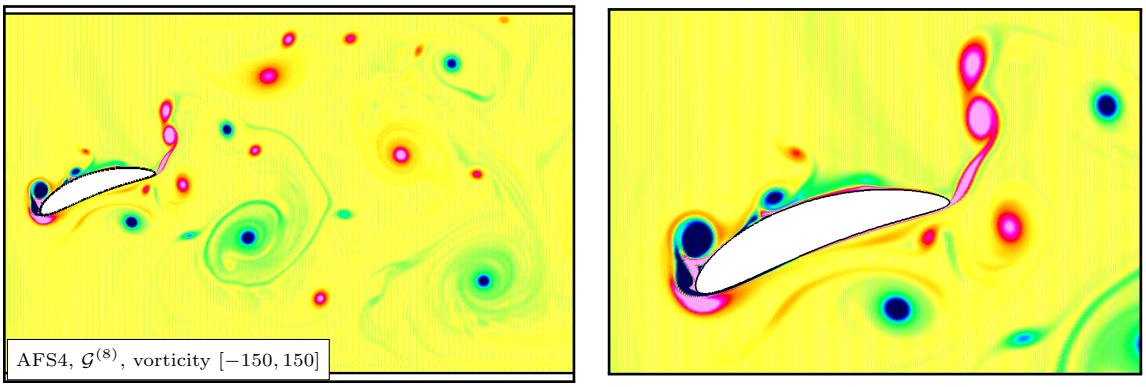

Figure 26: Flow past a pitching-plunging airfoil on grid $\mathcal{G}^{(8)}$. Contour plots of the vorticity at $t = 5$.

18.14 Flow past two spheres

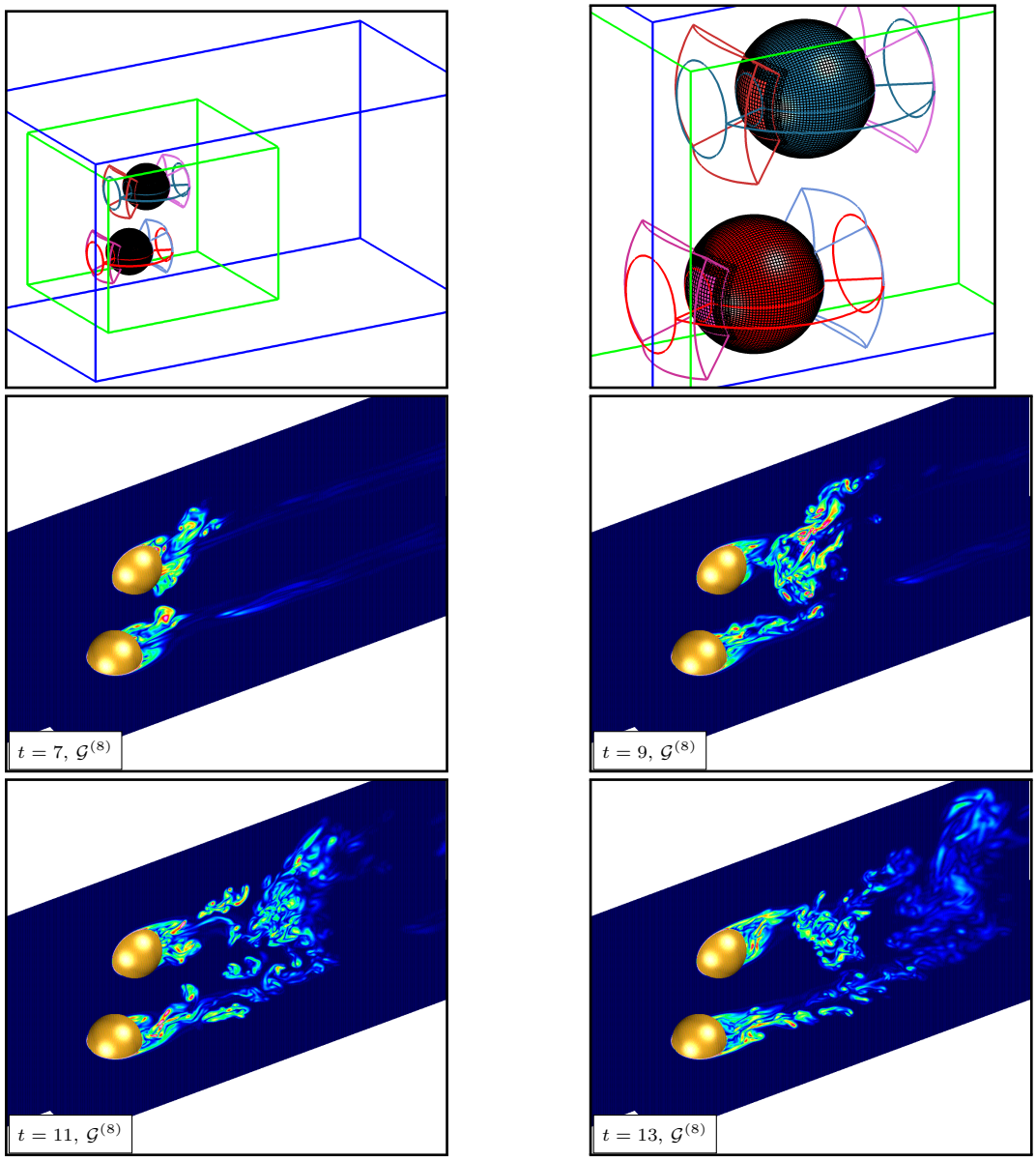

Figure 27: Flow past two spheres in channel. Top: Overlapping grid, $\mathcal{G}^{(3)}$ (fourth-order), for the spheres and channel. Contour plots of the enstrophy (max contour $\xi = 40$) on Grid $\mathcal{G}^{(8)}$ (41M pts) using the fourth-order accurate scheme AFS42.

We simulate the flow past two spheres in a channel. The grid for this problem was generated from the ogen command file `twoSpheresInAChannelGrid.cmd`. The solution was computed with the Cgins command file `cg/ins/cmd/sib.cmd`.

The geometry for the problem, as shown in Figure 27, consists of two spheres in a channel. The radius of each sphere is 0.5. They are each covered by three overlapping patches. The spheres are embedded in a refinement patch that extends a short distance into the wake. A coarser background grid covers the majority of the channel. Let $\mathcal{G}^{(j)}$ denote the composite grid for this geometry. The target grid spacing is $\Delta s = 1/(10j)$. The grid spacing is stretched in the normal direction to the body so that the boundary layer spacing is $\Delta s_{\rm bl}$.

Figure 27 shows the solution on grid $\mathcal{G}^{(8)}$. The incoming flow is in the x-direction with $u = 1$. Contours of the enstrophy ξ, (magnitude of the vorticity vector, $\xi = \|\nabla \times \mathbf{u}\|$) are shown. The solution was computed with the scheme AFS4 and the SSLES4 turbulence model ($\nu = 2 \times 10^{-5}$).

18.15 Flow past a deforming sphere

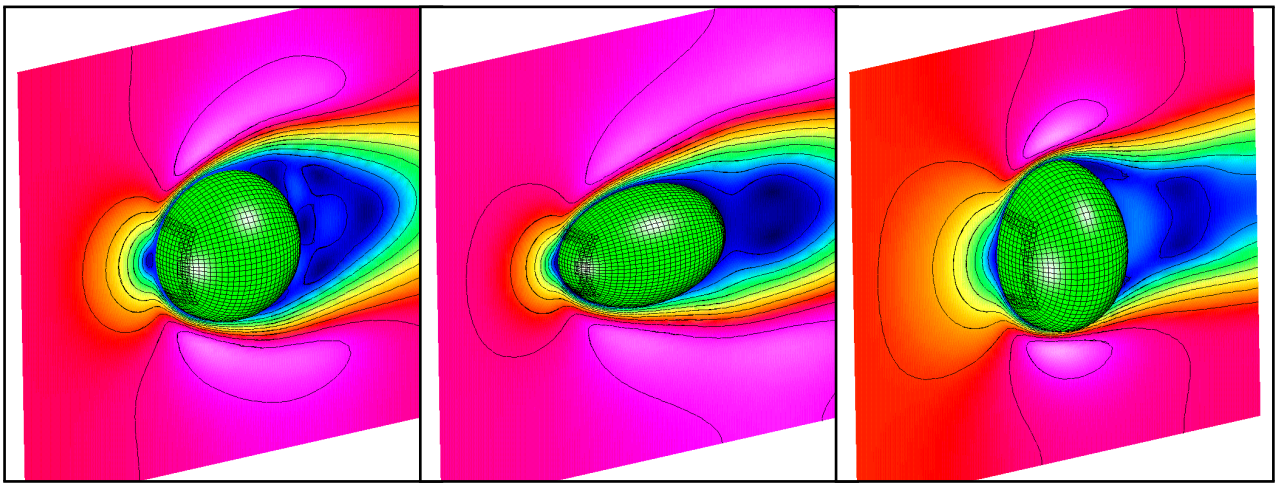

Figure 28: Flow past a deforming sphere.

In this section the flow past a deforming sphere is simulated. The grid for this problem was generated from the ogen command file `sibDeform.cmd`. The solution was computed with the Cgins command file `cg/ins/cmd/deform.cmd`.

The geometry for the problem, as shown in Figure 28, initially consists of a sphere of radius 0.5 in a channel. The sphere is covered by three overlapping patches. Let $\mathcal{G}^{(j)}$ denote the composite grid for this geometry. The target grid spacing is $\Delta s = 2/(10j)$.

The surface of the sphere is deformed over time into an ellipsoid and back again. The axes of the ellipse evolve according to

$$
\begin{aligned}
a(t) &= 1 + (a_0 - 1)\sin(\omega\pi t),\\
b(t) &= 1 + (b_0 - 1)\sin(\omega\pi t),\\
c(t) &= 1 + (c_0 - 1)\sin(\omega\pi t),
\end{aligned}
$$

where $a_0 = 0.8\alpha$ and $b_0 = c_0 = 1.2\alpha$ with $\alpha = 1$. The deformed surface is defined by

$$
\mathbf{x}(\mathbf{r}, t) = (a(t)x_0(\mathbf{r}), b(t)y_0(\mathbf{r}), c(t)z_0(\mathbf{r})),
$$

where the initial surface of the sphere is $(x_0(\mathbf{r}), y_0(\mathbf{r}), z_0(\mathbf{r}))$. The grids near the surface are recomputed at each time step using the hyperbolic grid generator.

Figure 28 shows the solution (magnitude of the velocity) at three times. The incoming flow is in the x-direction with $u = 1$.

18.16 Flow past a moving 3D cylinder at different Reynolds numbers

We simulate the flow past a moving three-dimensional cylinder using the second- and fourth-order accurate versions of the AFS scheme. The second-order and fourth-order SSLES turbulence models are used. The results indicate how the flow character changes as the effective Reynolds number increases.

The grid for this problem was generated from the ogen command file `cylinderInAChannel.cmd`. The solution was computed with the Cgins command file `cg/ins/cmd/cyl3d.cmd`.

The geometry for the problem consists of a cylinder of radius $r = 0.25$ in a channel $[-1,3]\times[-1,1]\times[-1,1]$. The axis of the cylinder is in the z-direction and the center of the cylinder is initially located at $(x,y) = (0,0)$. Let $\mathcal{G}^{(j)}$ denote the composite grid for this geometry. The target grid spacing is $\Delta s = 1/(10j)$. The grid spacing is stretched in the normal direction to the cylinder so that the boundary layer spacing is $\Delta s_{\rm bl}$.

The incoming flow is in the x-direction with $u = 1$.

Figure 29 shows the solution for a cylinder that oscillates up and down in the y-direction with a sinusoidal motion given by

$$y(t) = a_0 \sin(2\pi f_0 t),$$

where the amplitude is taken as $a_0 = 0.25$ and the frequency is $f_0 = 0.25$. The grid is $\mathcal{G}^{(2)}$ and the boundary layer spacing is 5 times smaller than the target grid spacing, with $\Delta s_{\rm bl}/\Delta s = 1/5$. Contours of the enstrophy ξ, (magnitude of the vorticity vector, $\xi = \|\nabla\times\mathbf{u}\|$) are shown. The solution was computed with the scheme AFS4 and the SSLES4 turbulence model ($\nu = 10^{-3}$). The results show that the flow develops into a fully three-dimensional wake with vortices being shed.

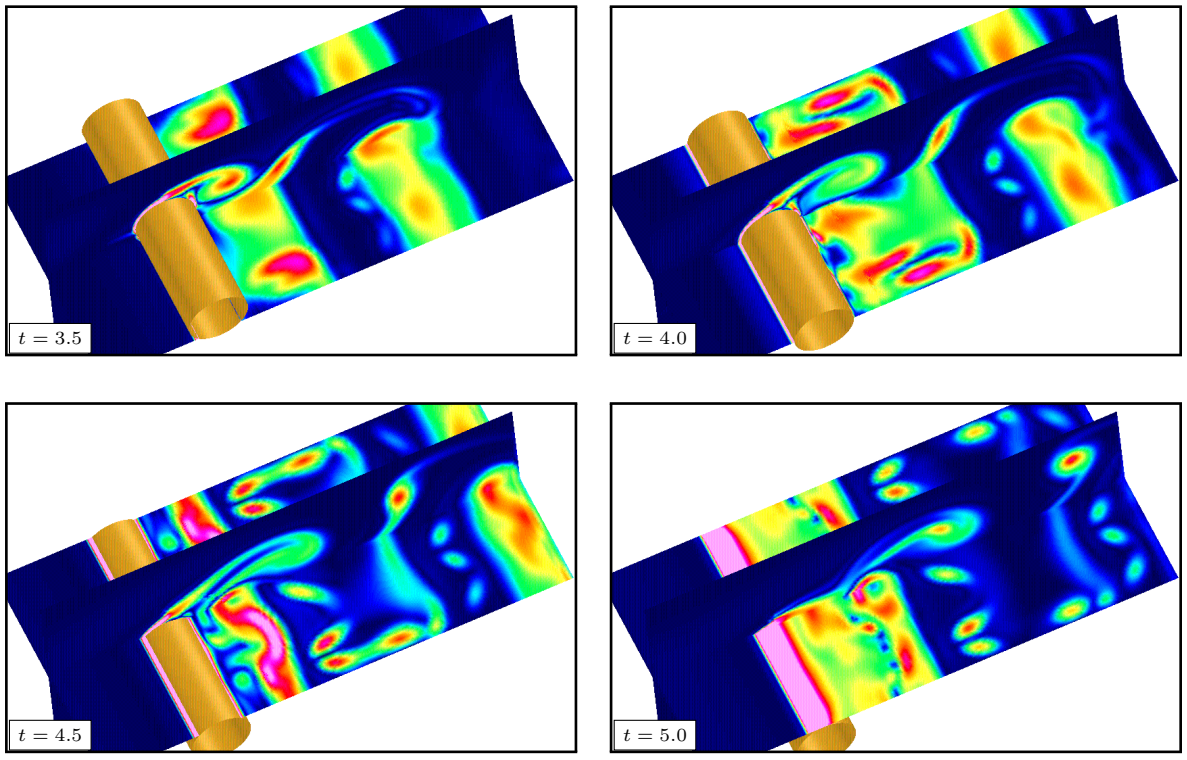

Figure 29: Flow past a 3D oscillating cylinder using Scheme AFS4 on grid $\mathcal{G}^{(2)}$. Contour plots of the enstrophy times $t = 3.5$, 4.0, 4.5 and 5.0. The contour levels are scaled to $[0,20]$. The cylinder is moving the $y-$*direction*.

18.17 Flow past a moving sphere

We simulate the flow past a moving sphere.

The grid for this problem was generated from the ogen command file `sphereInABox.cmd`. The solution was computed with the Cgins command file `cg/ins/cmd/sphereMove.cmd`.

The geometry for the problem consists of a sphere of radius $r = 1.0$, centered at the origin, in a channel $[-3,6]\times[-3,3]\times[-3,3]$. Let $\mathcal{G}^{(j)}$ denote the composite grid for this geometry. The target grid spacing is $\Delta s = 1/(10j)$. The grid spacing is stretched in the normal direction to the cylinder so that the boundary layer spacing is $\Delta s_{\rm bl}$.

Figure 30 shows the solution for a sphere that oscillates up and down in the y-direction with a sinusoidal motion given by

$$y(t) = a_0 \sin(2\pi f_0 t),$$

where the amplitude is taken as $a_0 = 0.25$ and the frequency is $f_0 = 0.5$. The incoming flow is in the x-direction with $u = 1$.

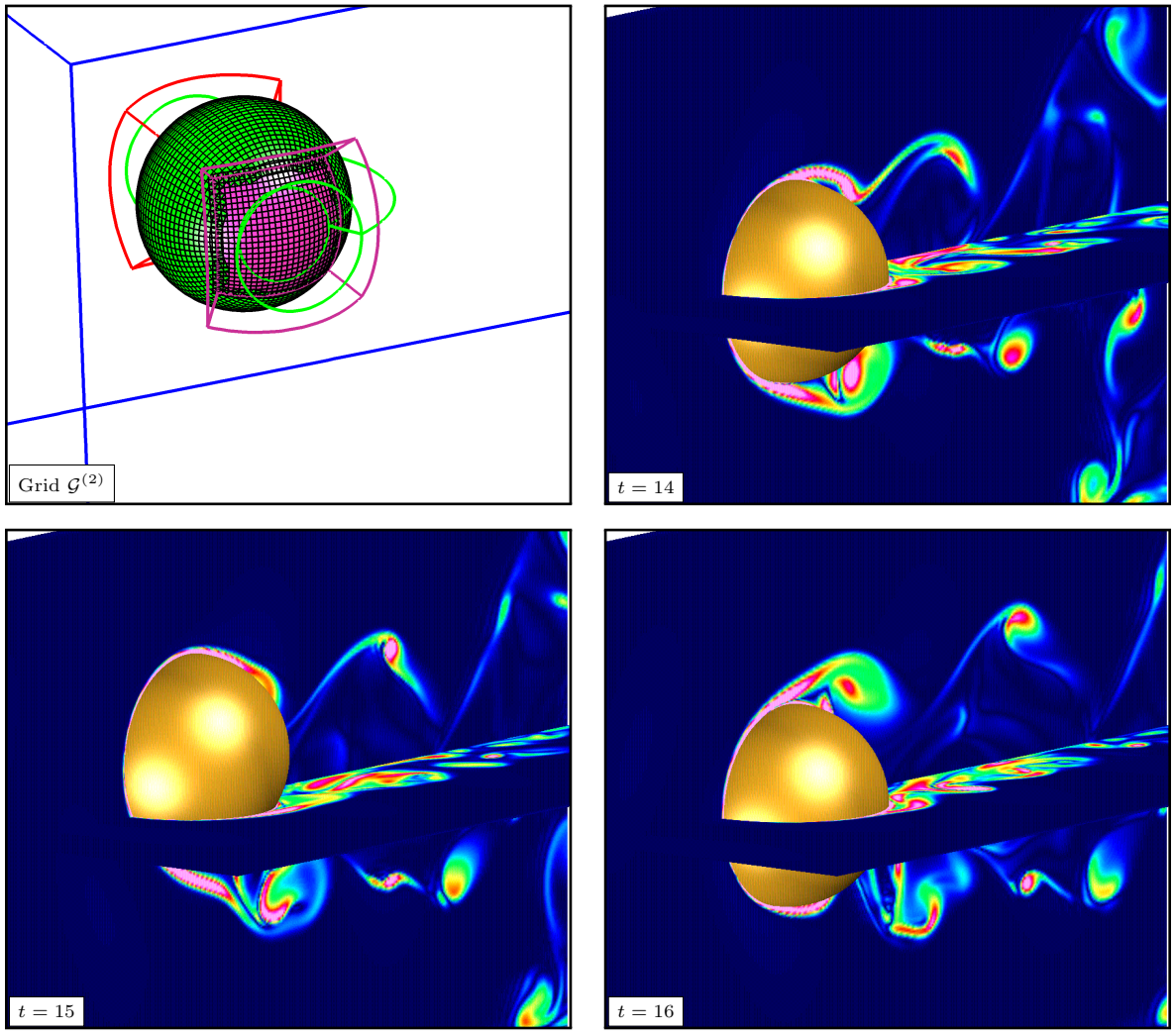

Figure 30: Flow past an oscillating sphere using scheme IM2 on grid $\mathcal{G}^{(4)}$ with $\nu = 2\times10^{-4}$. Contour plots of the enstrophy. The contour levels are scaled to $[0, ??]$. The cylinder is moving the $y-$ *direction*.

18.18 Flow past a cube in a channel

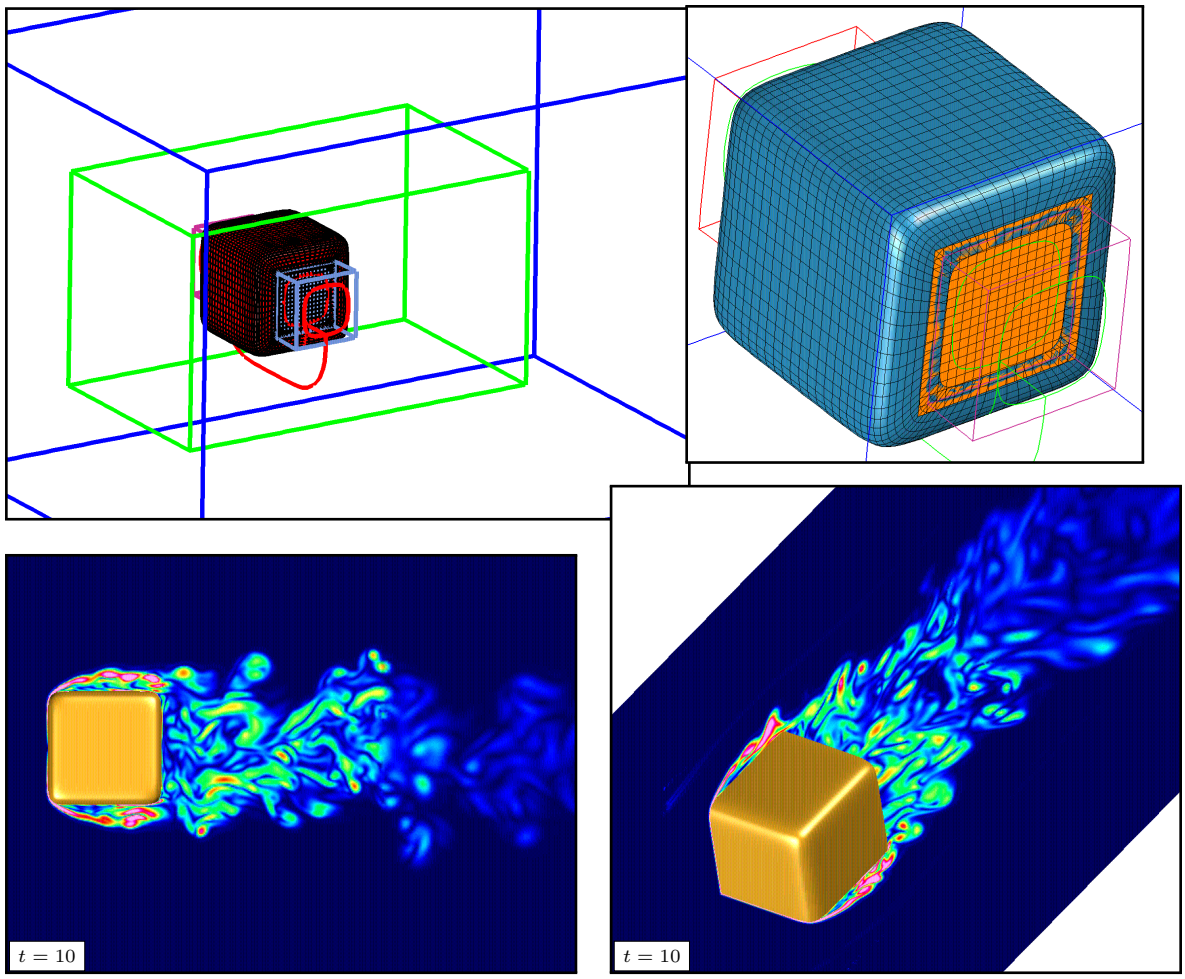

Figure 31: Flow past a cube in a channel. Top: Overlapping grid for the cube and channel. Bottom: contour plots of the enstrophy from a simulation on grid $\mathcal{G}^{(8)}$ (14M grid points), using the scheme AFS42. Evident is the transition of the enstrophy midway in the wake onto the coarser background grid.

We simulate the flow past the exterior of a cube (with rounded corners).

The grid for this problem was generated from the ogen command file `loftedBox.cmd`. The solution was computed with the Cgins command file `cg/ins/cmd/boxInAChannel.cmd`.

The geometry for the problem consists of a cube with sides of length one in a channel $[-1, 2.5] \times [-2, 2] \times [-2, 2]$. Let $\mathcal{G}^{(j)}$ denote the composite grid for this geometry. The target grid spacing is $\Delta s = 1/(10j)$. The grid spacing is stretched in the normal direction to the box so that the boundary layer spacing is $\Delta s_{\rm bl}$. The grid is shown in Figure 31. The grid for the cube consists three grids, a main patch covering four faces and all edges and two end *caps*. This grid was generated with the *LoftedSurfaceMapping*. We note that is quite difficult to make a high quality grid for this apparently simple geometry since the edges and corners must be sufficiently resolved. The cube grid are embedded in a Cartesian refinement grid (block boundaries shown in green), which is in turn located in a larger background Cartesian grid with grid spacing $2\Delta s$.

The incoming flow is in the x-direction with $u = 1$. Figure 31 shows the solution computed on grid $\mathcal{G}^{(8)}$, which had a total of approximately 14M grid points. Contours of the enstrophy ξ, (magnitude of the vorticity vector, $\xi = \|\nabla \times \mathbf{u}\|$) are shown. The solution was computed with the scheme AFS4 and the SSLES4 turbulence model ($\nu = 2 \times 10^{-5}$).

Figure 32 shows results from flow past a rotating cube.

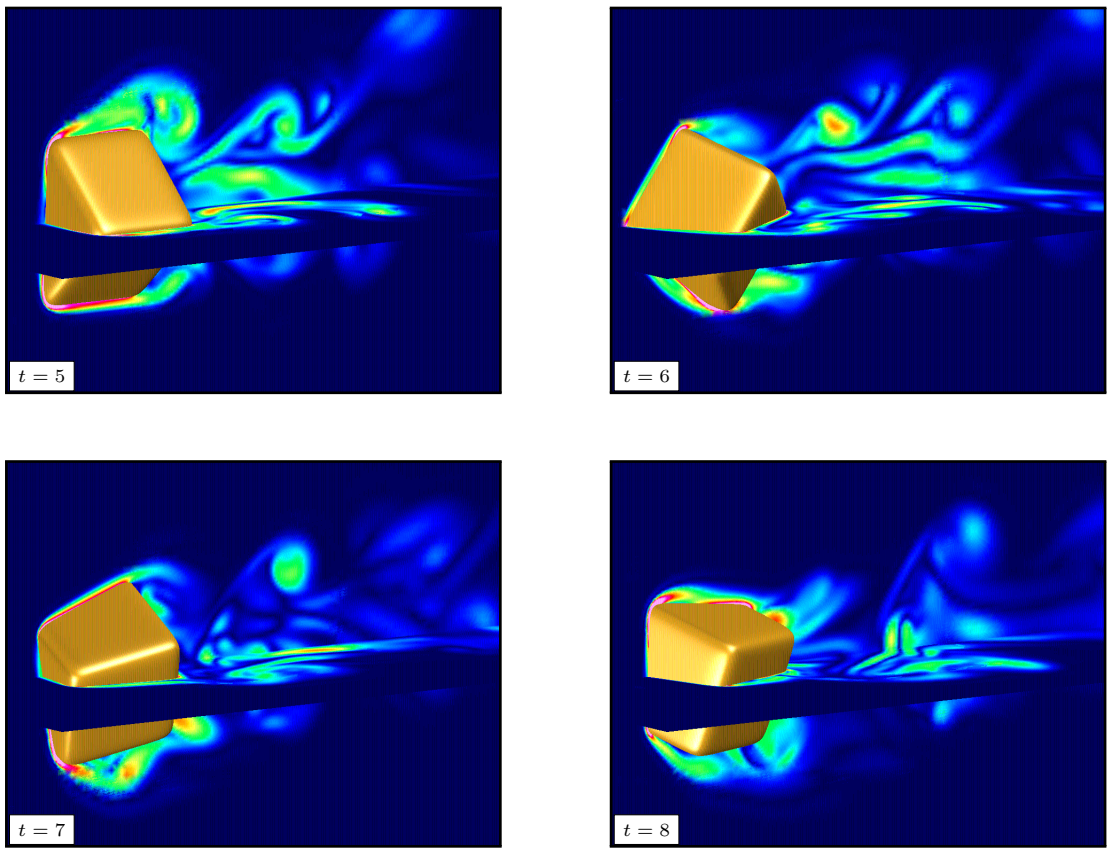

Figure 32: Flow past a rotating cube in a channel. Contour plots of the enstrophy from a simulation on grid $\mathcal{G}^{(4)}$, using the scheme AFS42.

18.19 Flow past a rotating flattened torus

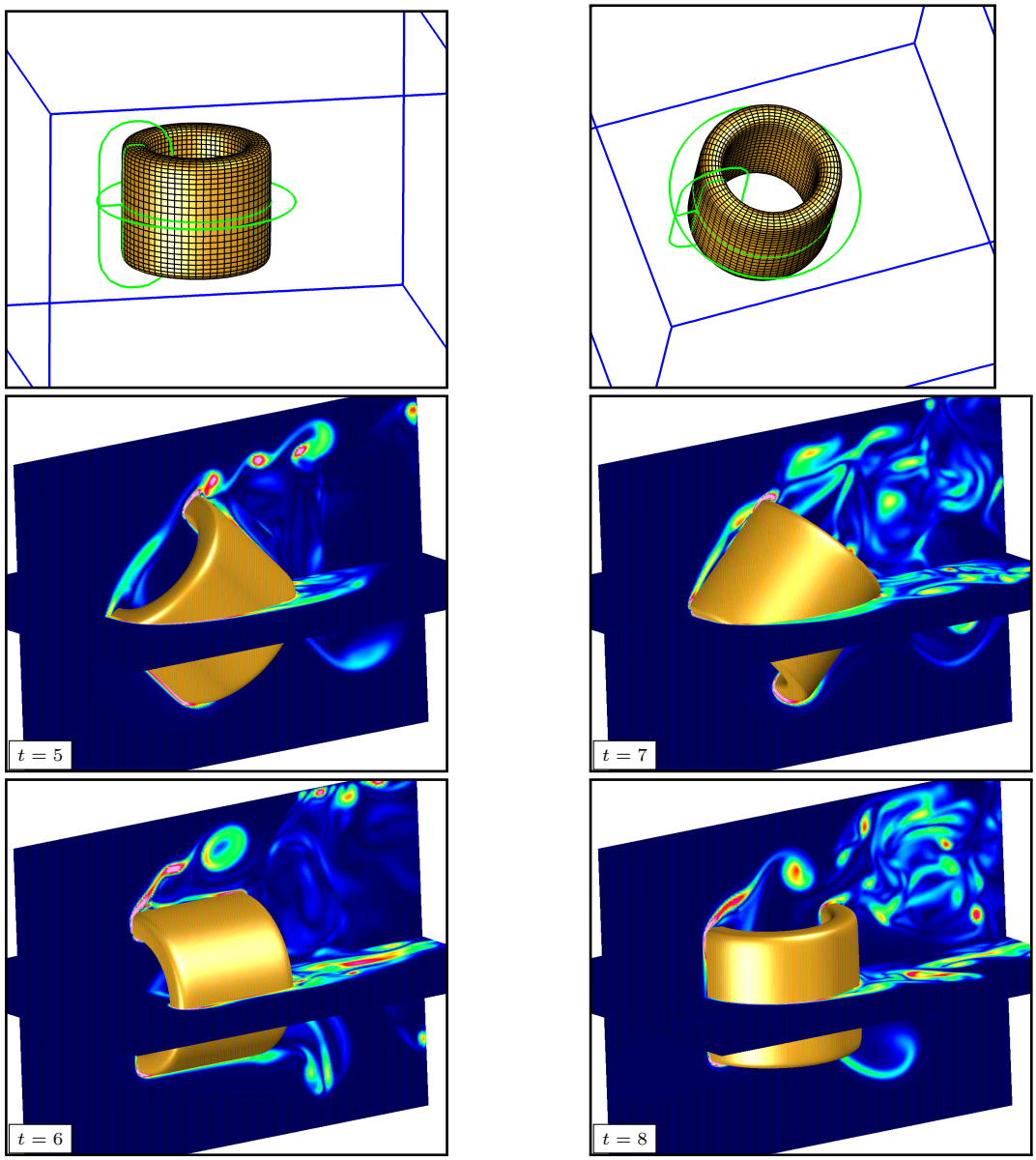

Figure 33: Flow past a rotating *flattened torus* in channel. Top: Overlapping grid for the torus and channel. Contour plots of the enstrophy on grid $\mathcal{G}^{(8)}$ (14M grid points), using scheme AFS42.

We simulate the flow past a moving *torus* (the cross section of the torus being a smoothed thin rectangle). The grid for this problem was generated from the ogen command file `flattenedTorusGrid.cmd`. The solution was computed with the Cgins command file `cg/ins/cmd/flattenedTorus.cmd`.

The geometry for the problem, as shown in Figure 33, consists of toroidal shaped body of revolution with cross-section defined by a smoothed polygon for the rectangle $[-.05,.05]\times[-.5,-5]$. This cross-section is rotated about the line through the point $(0,.5,0)$ with tangent along the y-axis. This toridal ring is placed in a channel with dimensions ??. Let $\mathcal{G}^{(j)}$ denote the composite grid for this geometry. The target grid spacing is $\Delta s = 1/(10j)$. The grid spacing is stretched in the normal direction to the box so that the boundary layer spacing is $\Delta s_{\rm bl}$.

The incoming flow is in the x-direction with $u=1$. Figure 33 shows the solution for ... Contours of the enstrophy ξ, (magnitude of the vorticity vector, $\xi = \|\nabla\times\mathbf{u}\|$) are shown. The solution was computed with the scheme AFS4 and the SSLES4 turbulence model ($\nu = 10^{??}$).

18.20 Flow past a rotating disk

We simulate the flow past a rotating *disk* ("pill" or "coin"). The grid for this problem was generated from the ogen command file `pillInABoxGrid.cmd`. The solution was computed with the Cgins command file `cg/ins/cmd/pillInABox.cmd`. The geometry for the problem, as shown in Figure 34, consists of disk ("pill") shaped body. The edge of the disk is constructed from a body of revolution using a cross-section defined by a smooth-polygon. The disk is embedded in a channel. A refinement patch is located near the disk. A coarser background grid covers the majority of the channel. Let $\mathcal{G}^{(j)}$ denote the composite grid for this geometry. The target grid spacing is $\Delta s = 1/(10j)$. The grid spacing is stretched in the normal direction to the body so that the boundary layer spacing is $\Delta s_{\rm bl}$.

Figure 34 shows the solution on grid $\mathcal{G}^{(8)}$ (7M pts). The incoming flow is in the x-direction with $u = 1$. Contours of the enstrophy ξ, (magnitude of the vorticity vector, $\xi = \|\nabla \times \mathbf{u}\|$) are shown with the maximum contour level set at $\xi = 40$. The solution was computed with the scheme AFS4 and the SSLES4 turbulence model ($\nu = 10^{-4}$).

Figure 35 shows the solution on the finer grid $\mathcal{G}^{(16)}$ (51M pts).

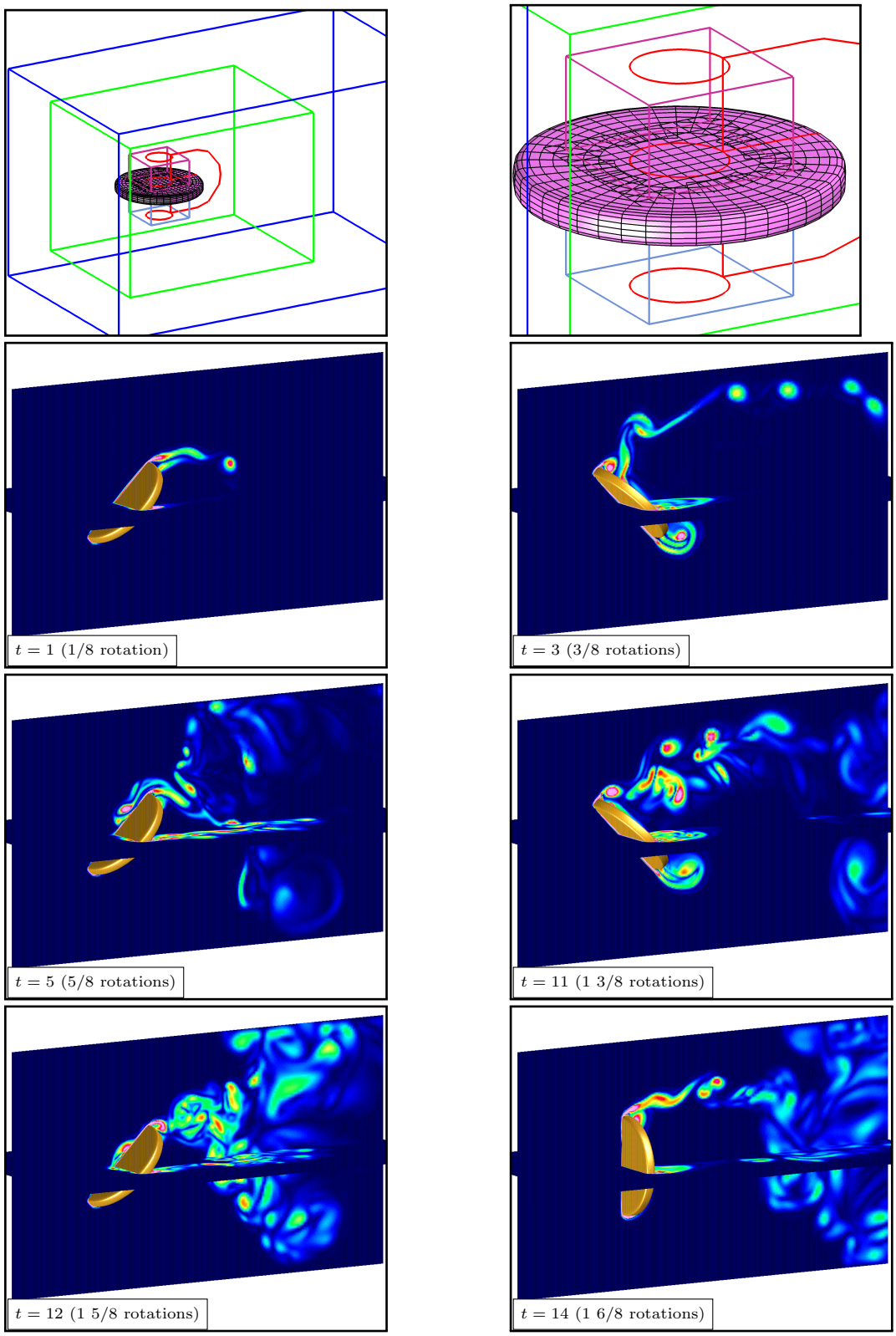

Figure 34: Flow past a rotating *disk* in channel. Top: Overlapping grid, $\mathcal{G}^{(2)}$ (second-order), for the disk and channel. Contour plots of the enstrophy (max contour $\xi = 40$) on grid $\mathcal{G}^{(8)}$, using the fourth-order accurate scheme AFS42.

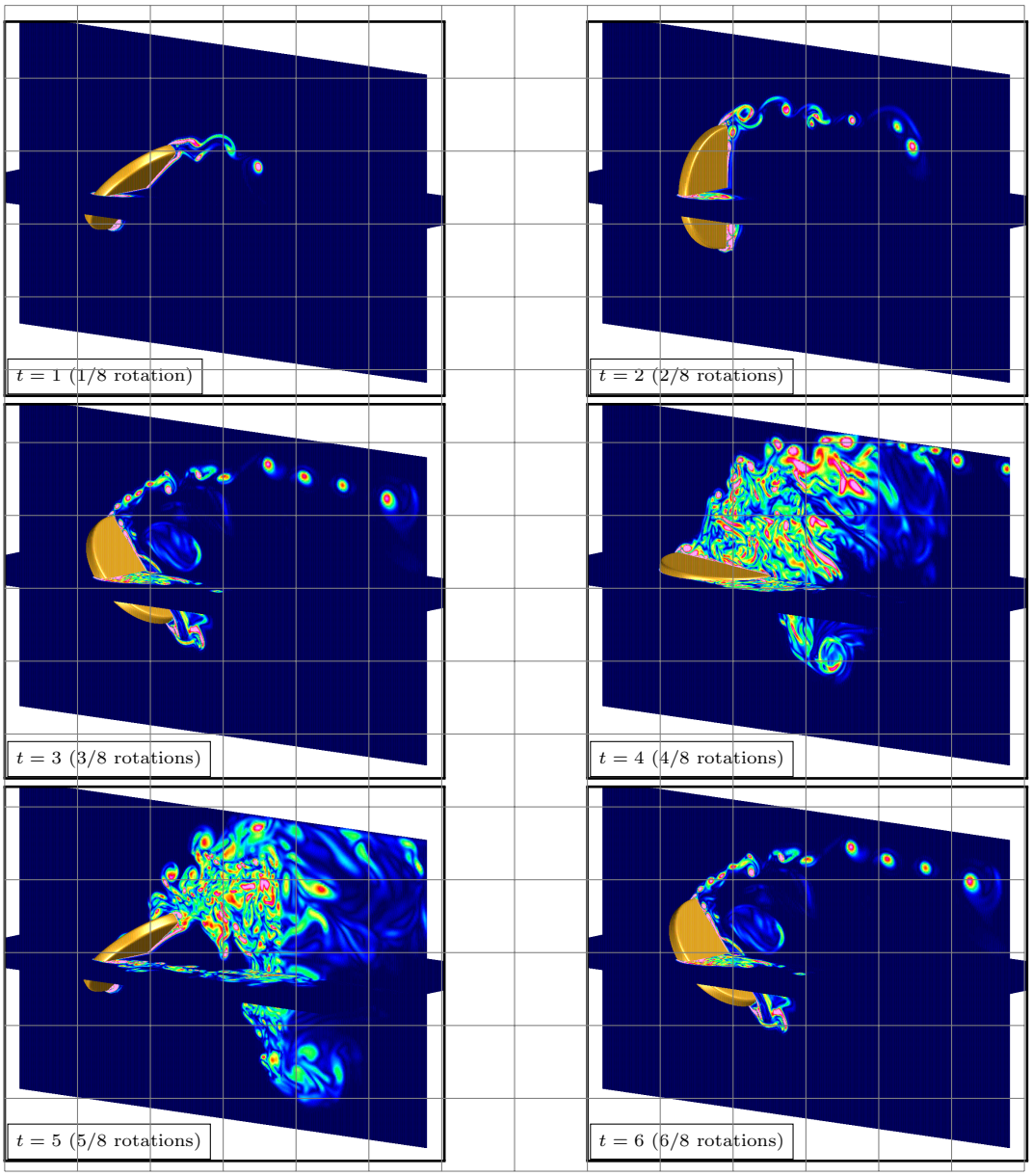

Figure 35: FINISH ME. Flow past a rotating *disk* in channel. Contour plots of the enstrophy (max contour $\xi = 50$) on grid $\mathcal{G}^{(16)}$, using the fourth-order accurate scheme AFS42.

18.21 Flow past a rounded blade

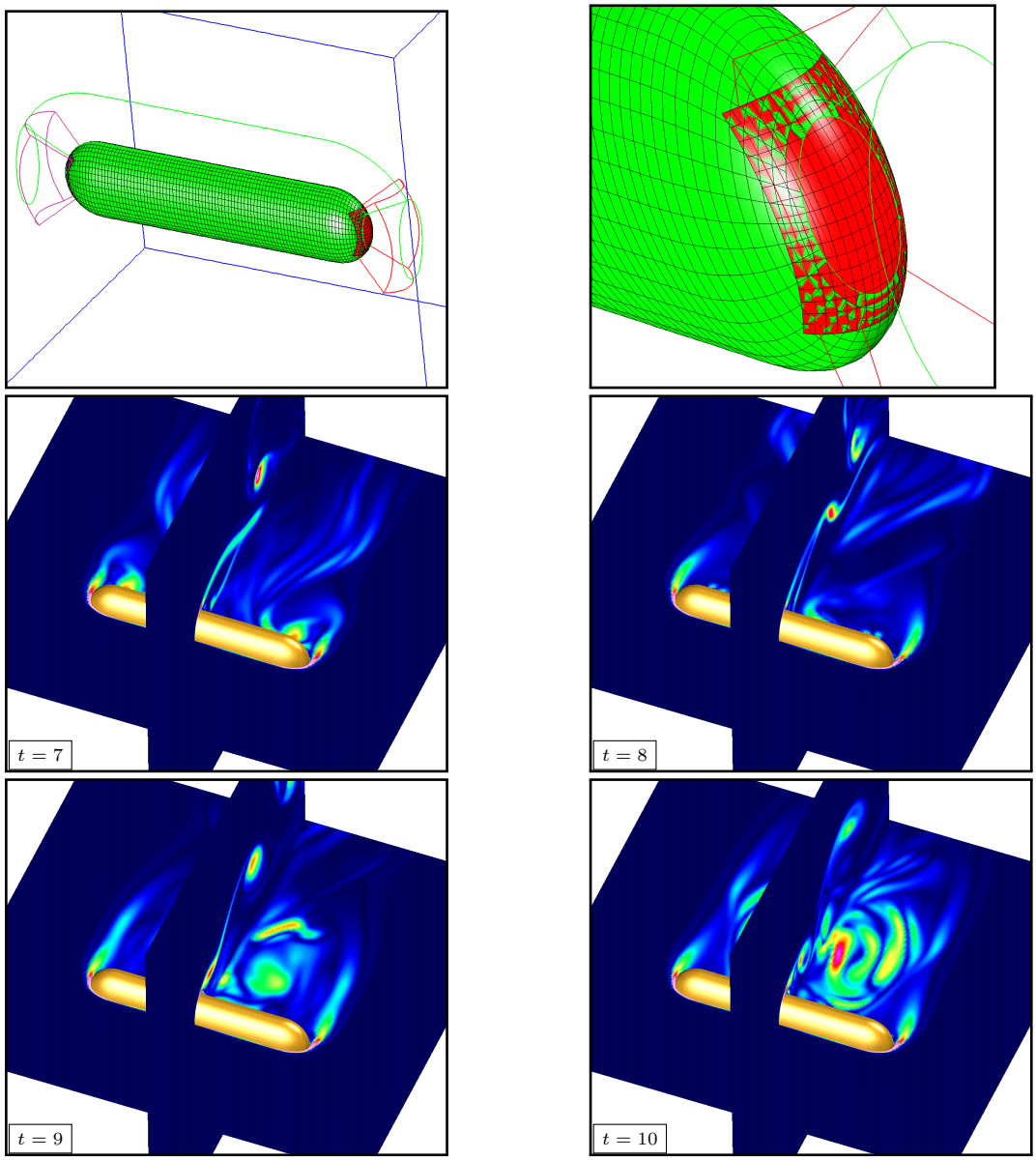

Figure 36: Flow past a *rounded blade* in channel. Top: Overlapping grid for the blade and channel. Contour plots of the enstrophy on grid $\mathcal{G}^{(4)}$ (7M pts), using scheme AFS42.

We simulate the flow past a stationary and moving *rounded blade*.

The grid for this problem was generated from the ogen command file `roundedBladeGrid.cmd`. The solution was computed with the Cgins command file `cg/ins/cmd/roundedBlade.cmd`.

The geometry for the problem, as shown in Figure 36, consists of flattened cylinder with rounded ends. Let $\mathcal{G}^{(j)}$ denote the composite grid for this geometry. The target grid spacing is $\Delta s = 1/(10j)$. The grid spacing is stretched in the normal direction to the box so that the boundary layer spacing is $\Delta s_{\rm bl}$.

The incoming flow is in the y-direction with $v = 1$.

Figure 36 shows the solution for a non-rotating blade while Figure 37 shows results for a rotating blade (one full rotation takes 4 seconds). Contours of the enstrophy ξ, (magnitude of the vorticity vector, $\xi = \|\nabla \times \mathbf{u}\|$) are shown. The solution was computed with the scheme AFS4 and the SSLES4 turbulence model ($\nu = 10^{??}$).

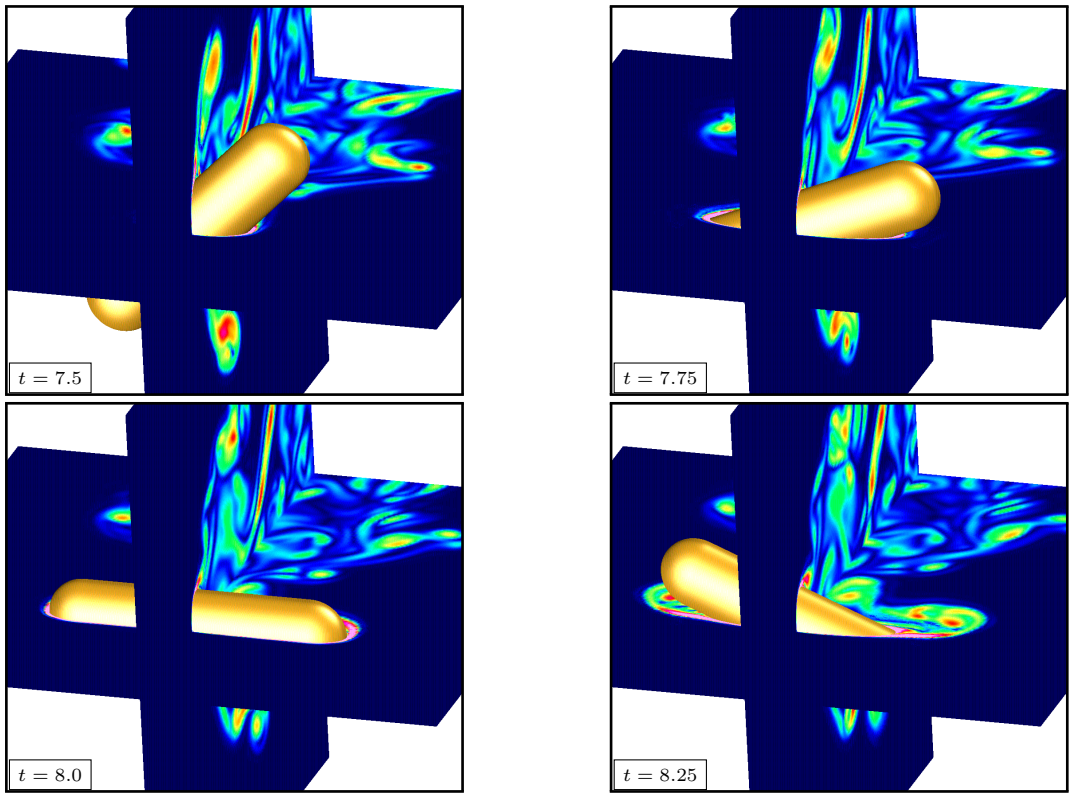

Figure 37: Flow past a rotating *rounded blade* in channel. Contour plots of the enstrophy on grid $\mathcal{G}^{(4)}$ (7M pts), using scheme AFS42.

18.22 Wind turbine and tower

As a demonstration, we simulate the flow past a model of a wind turbine. The turbine consists of a stationary tower and three rotating blades.

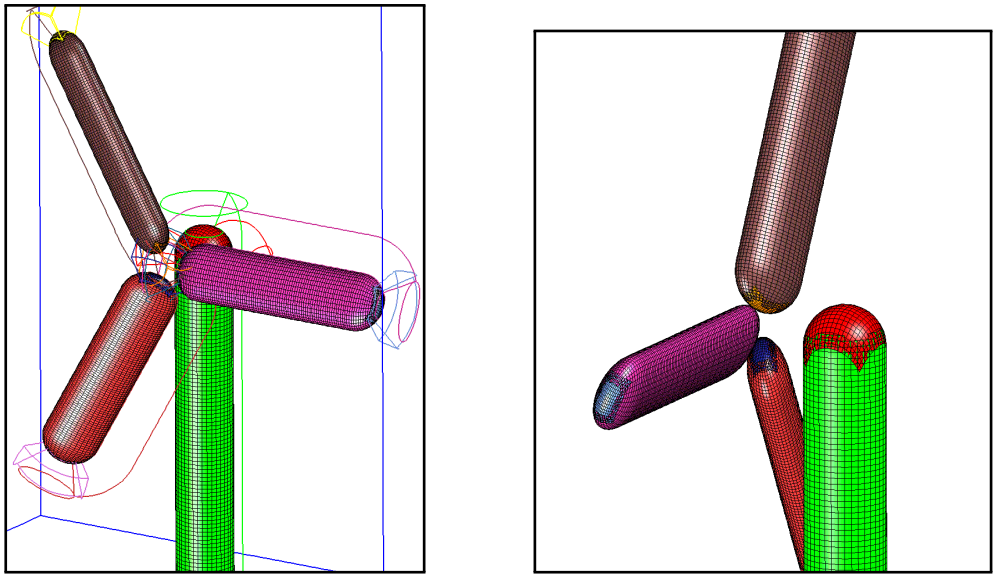

Figure 38: Overlapping grid for a model wind turbine consisting of a tower and three blades.

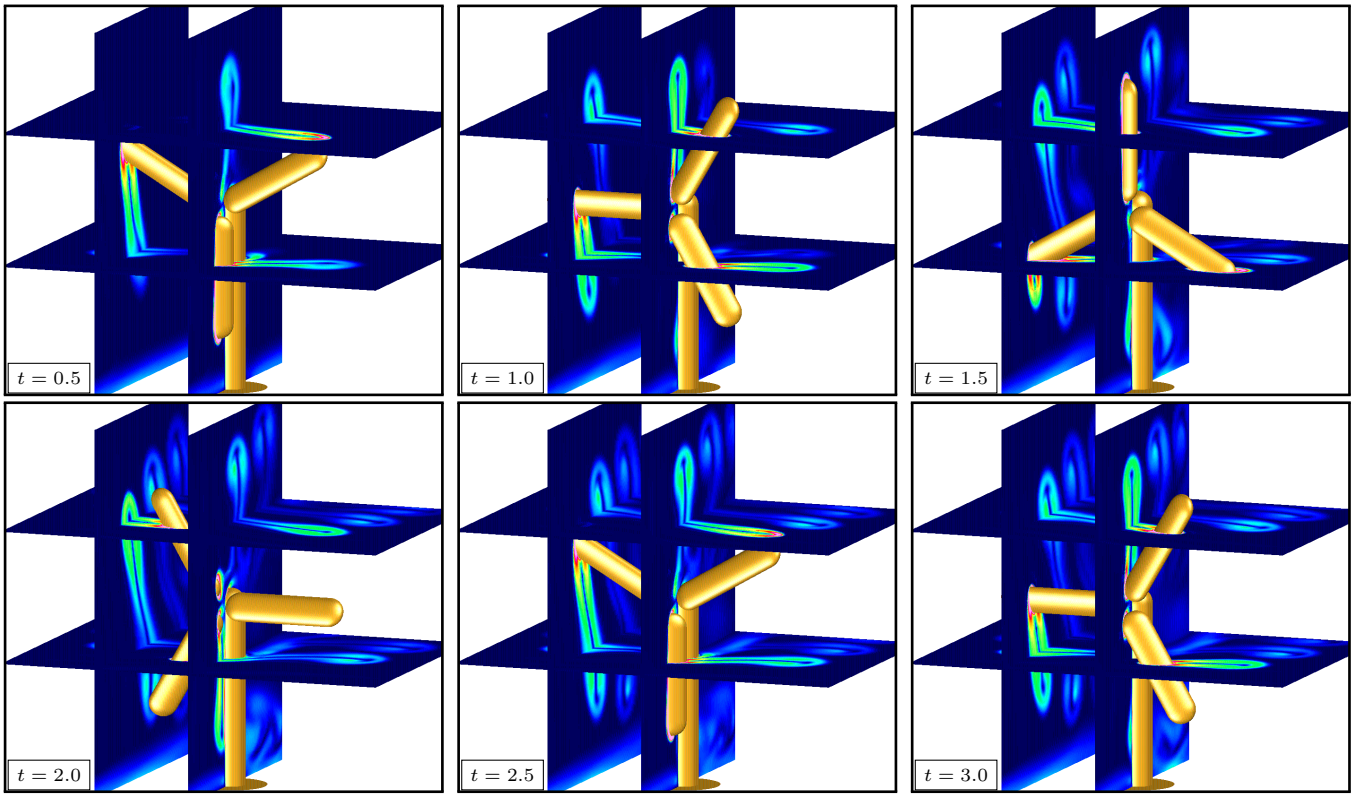

Figure 39: Flow past a wind turbine with rotating blades. The blades rotated in a clockwise direction and make one full revolution every 2 time units. Contour plots of the enstrophy on grid $\mathcal{G}^{(2)}$ (2.5M pts), using scheme AFS22.

The grid for this problem was generated from the ogen command file `turbineAndTower.cmd`. The solution was computed with the Cgins command file `cg/ins/cmd/turbineAndTower.cmd`.

The geometry for the problem, as shown in Figure 38, consists of tower and three blades. Let $\mathcal{G}^{(j)}$ denote the composite grid for this geometry. The target grid spacing is $\Delta s = 1/(10j)$. The grid spacing is stretched in the

normal direction to the box so that the boundary layer spacing is Δs_{bl}.

The incoming flow is in the y-direction with $v = 2$.

Figure 39 shows the solution on the grid $\mathcal{G}^{(2)}$ (2.5M pts), using scheme AFS22. Contours of the enstrophy ξ, (magnitude of the vorticity vector, $\xi = \|\nabla \times \mathbf{u}\|$) are shown.

Figure 40 shows the solution on the grid $\mathcal{G}^{(2)}$ (2.5M pts), using the fourth-order accurate scheme AFS42. Contours of the enstrophy ξ, (magnitude of the vorticity vector, $\xi = \|\nabla \times \mathbf{u}\|$) are shown.

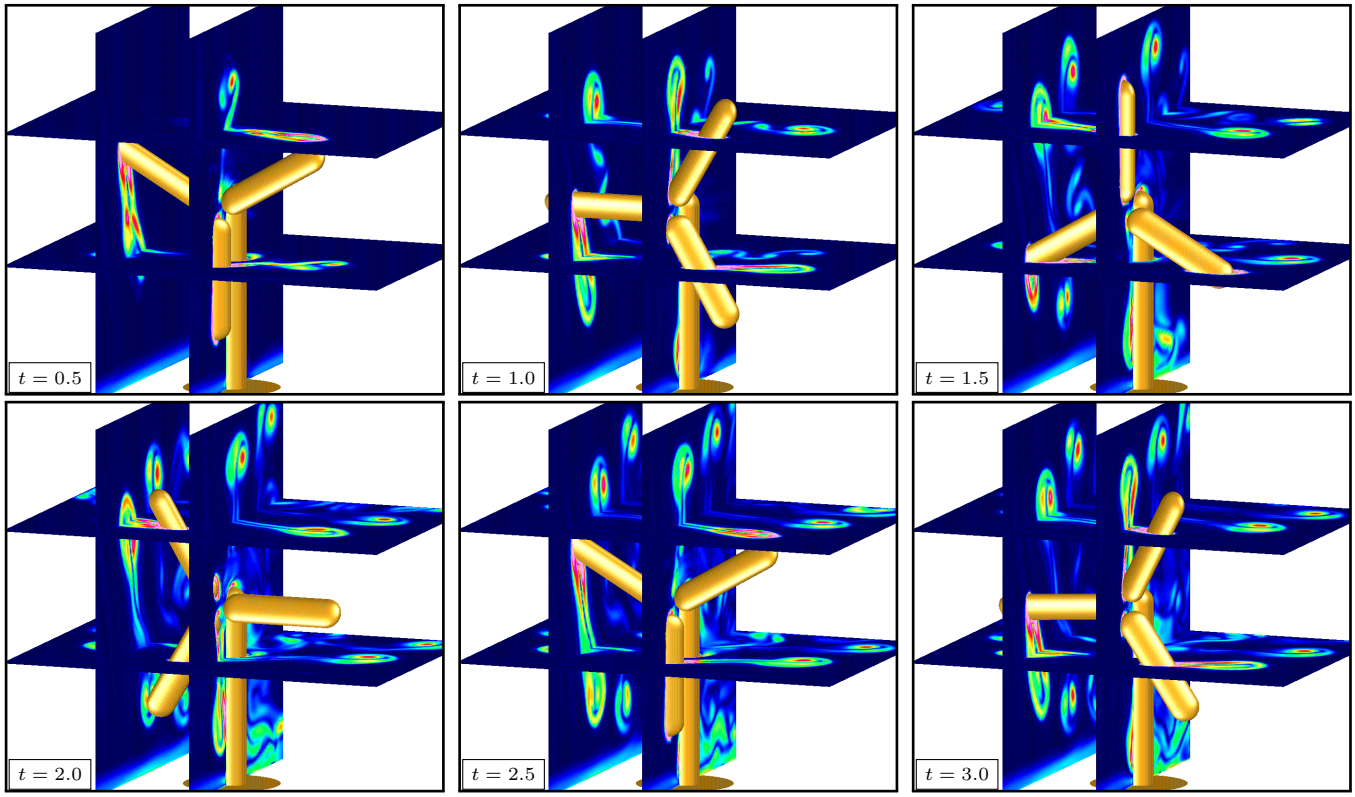

Figure 40: Flow past a wind turbine with rotating blades. The blades rotated in a clockwise direction and make one full revolution every 2 time units. Contour plots of the enstrophy on grid $\mathcal{G}^{(2)}$ (2.5M pts), using the fourth-order accurate scheme AFS42.

18.23 Flow over terrain

To run this example see `cg/ins/runs/terrain/Readme`.

We consider the flow over a terrain. The terrain elevation data is obtained from the USGS web site ...

The matlab script `convertGIS.m` is used to convert the GIS data into Cartesian coordinates (from lat-long). This matlab script can also be used to optionally smooth the terrain, choose a sub-region, or choose a two-dimensional cross section.

The ogen script `terrainGrid.cmd` takes the output from `convertGIS.m` and constructs an overlapping grid for the terrain.

Figure 41 shows a 2D and 3D grid and some preliminary solutions.

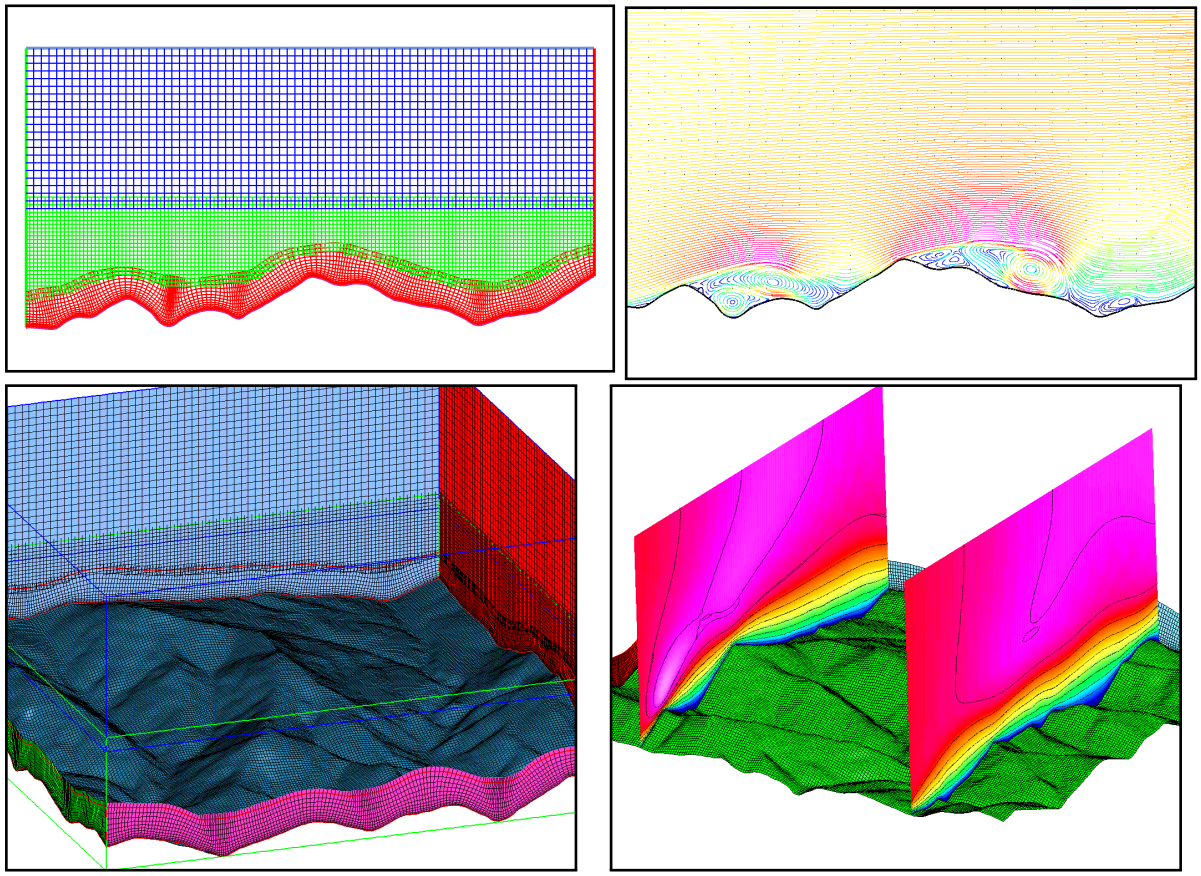

Figure 41: Flow over terrain.

Figure 42 shows some results from the fourth-order accurate scheme AFS24 on grid $\mathcal{G}_t^{(4)}$ (10M pts). The size the of the domain is approximately $1.5km \times 1.5km$ in the horizontal. The grid spacing near the surface is approximately $5m$ in the horizontal and $.5m$ for the grid cell next to the surface.

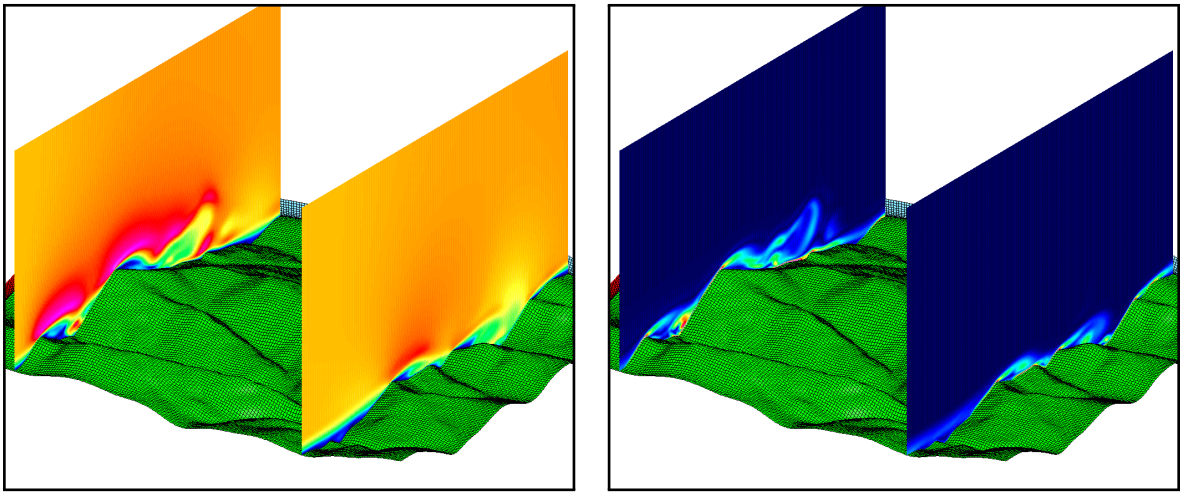

Figure 42: Flow over site300 terrain using grid $\mathcal{G}_t^{(4)}$ (10M pts) and scheme AFS24. Left: speed, right: enstrophy. The surface grid is coarsened by a factor of 2 for plotting purposes.

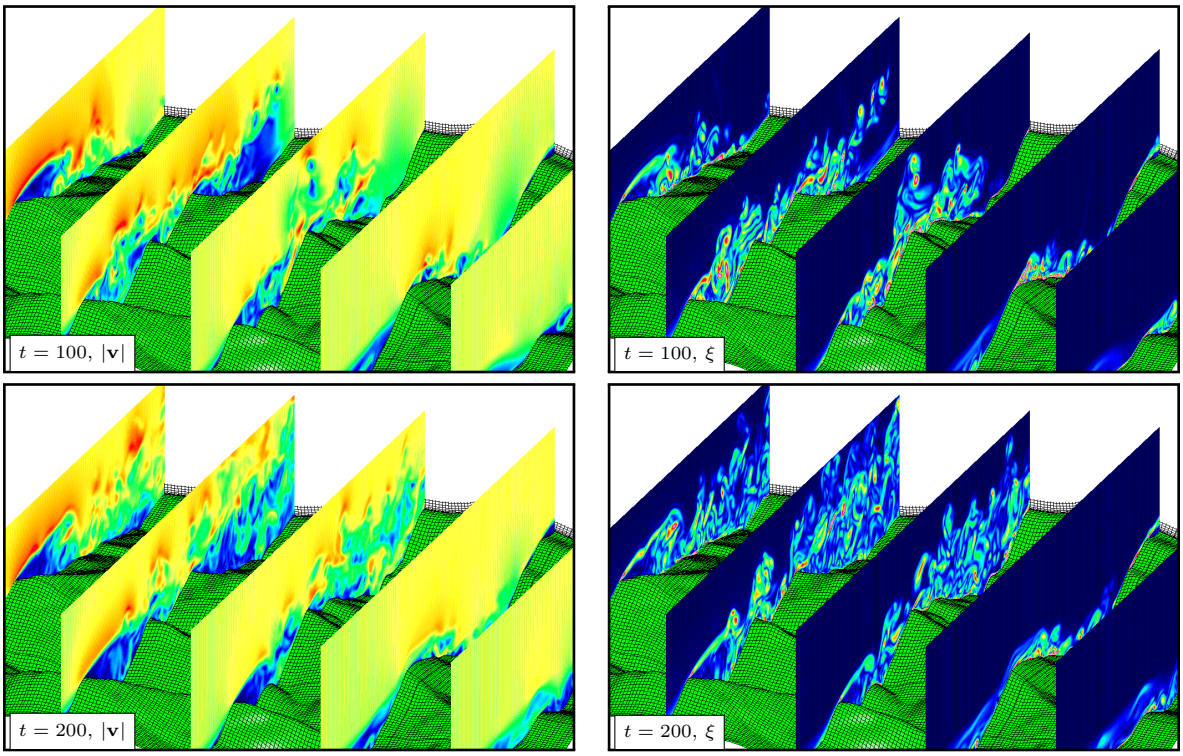

Figure 43: Flow over site300 terrain using grid $\mathcal{G}_t^{(8)}$ (60M pts) and scheme AFS24. The flow speed, $|\mathbf{v}|$ and enstrophy, ξ, ($\xi \in [0, 2]$), are shown. The contour planes do not extend to the full height of the computational domain. The surface grid is coarsened by a factor of 4 for plotting purposes.

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, p. .

[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] P. M. Gresho, *Some interesting issues in incompressible fluid dynamics, both in the continuum and in the numerical simulation*, Advances in Applied Mechanics, 28 (1992).

[4] W. D. Henshaw, *A fourth-order accurate method for the incompressible Navier-Stokes equations on overlapping grids*, J. Comput. Phys., 113 (1994), pp. 13–25. publications/icns1994.pdf.

[5] ——, *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. D. Knight, eds., 1996, pp. 281–290.

[6] ——, *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.

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

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

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

[10] ——, *Other stuff for Overture, user guide, version* 1.0, Research Report UCRL-MA-134292, Lawrence Livermore National Laboratory, 1999.

[11] ——, *Cgins user guide: An Overture solver for the incompressible Navier-Stokes equations on composite overlapping grids*, Software Manual LLNL-SM-455851, Lawrence Livermore National Laboratory, 2010. publications/CginsUserGuide.pdf.

[12] W. D. Henshaw and H.-O. Kreiss, *Analysis of a difference approximation for the incompressible Navier-Stokes equations*, Research Report LA-UR-95-3536, Los Alamos National Laboratory, 1995.

[13] W. D. Henshaw, H.-O. Kreiss, and L. G. M. Reyna, *On the smallest scale for the incompressible Navier-Stokes equations*, Theoretical and Computational Fluid Dynamics, 1 (1989), pp. 65–95.

[14] ——, *Smallest scale estimates for the incompressible Navier-Stokes equations*, Arch. Rational Mech. Anal., 112 (1990), pp. 21–44.

[15] ——, *A fourth-order accurate difference approximation for the incompressible Navier-Stokes equations*, Comput. Fluids, 23 (1994), pp. 575–593. publications/HenshawKreissReynaFouthOrderINS1994.pdf.

[16] J. D. Lambert, *Computational Methods in Ordinary Differential Equations*, Cambridge University Press, New York, 1973.

[17] D. C. Wilcox, *Turbulence Modeling for CFD*, DCW Industries, 2000.

Index