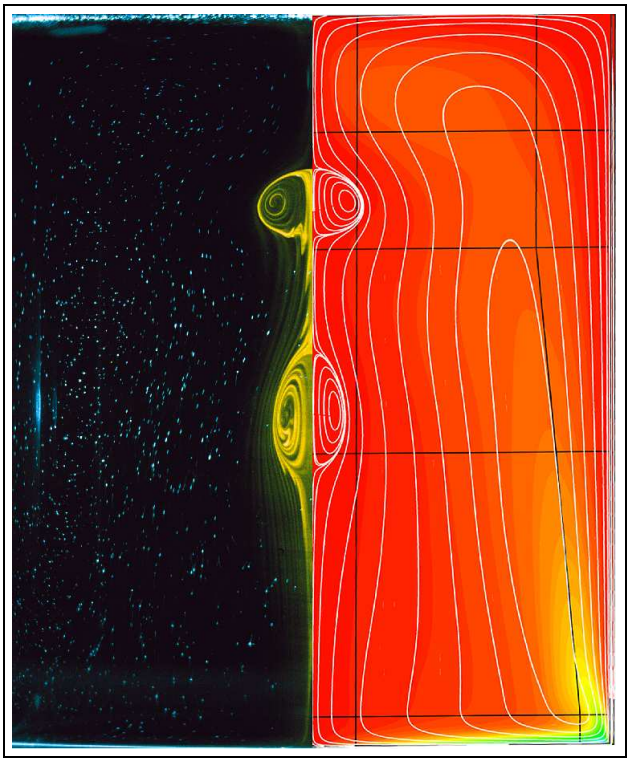
Using *Semtex*



H. M. Blackburn
Monash University

July 15, 2007
Semtex version 6.1

Contents

Chapter 1

Introduction

Semtex is a family of spectral element simulation codes. The spectral element method is a high-order finite element technique that combines the geometric flexibility of finite elements with the high accuracy of spectral methods. The method was pioneered in the mid 1980's by Anthony Patera at MIT (Patera; 1984; Korczak and Patera; 1986). *Semtex* uses parametrically mapped quadrilateral elements, the classic GLL 'nodal' shape function basis, and continuous Galerkin projection. Algorithmically the code is similar to Ron Henderson's *Prism* (Henderson and Karniadakis; 1995; Karniadakis and Henderson; 1998; Henderson; 1999), but with some differences in design, and lacking mortar element capability. A notable extension is that *Semtex* can solve problems in cylindrical as well as Cartesian coordinate systems (Blackburn and Sherwin; 2004).

1.1 Numerical method

Some central features of the spectral element method are

Orthogonal polynomial-based shape functions Spectral accuracy is achieved by using tensor-product Lagrange interpolants within each element, where the nodes of these shape functions are placed at the zeros of Legendre polynomials mapped from the canonical domain [-1, 1]×[-1, 1] to each element. In one spatial dimension, the resulting Gauss–Lobatto–Legendre Lagrange interpolant which is unity at one of the $N+1$ Gauss–Lobatto points x_j in [-1, 1] and zero at the others is

$$\psi_j(x) = \frac{1}{N(N+1)L_N(x_j)} \frac{(1-x^2)L'_N(x)}{x-x_j}. \tag{1.1}$$

For example, the family of sixth-order GLL Lagrange interpolants is shown in figure 1.1. In smooth function spaces it can be shown that the resulting interpolants converge exponentially fast (faster than any negative integer power of N) as the order of the interpolant is increased. See Canuto et al. (1988), §§ 2.3.2 and 9.4.3.

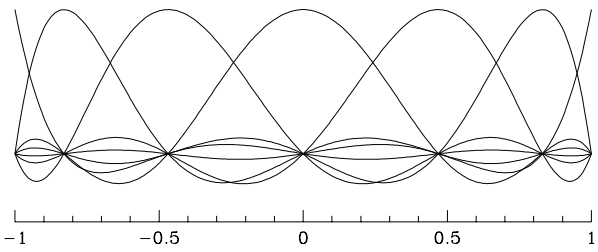


Figure 1.1: The family of sixth-order one-dimensional GLL Lagrange shape functions on the master domain $[-1, +1]$.

Gauss–Lobatto quadrature Efficiency (particularly in iterative methods) is achieved by using Gauss–Lobatto quadrature for evaluating elemental integrals: the quadrature points reside at the nodal points, which enables fast tensor-product techniques to be used for iterative matrix solution methods. Gauss–Lobatto quadrature delivers diagonal mass matrices.

Static condensation Direct matrix solutions are sped up by using static condensation coupled with bandwidth reduction algorithms to reduce storage requirements for assembled system matrices.

While the numerical method is very accurate and efficient, it also has the advantage that complex geometries can be accommodated by employing unstructured meshes. The vertices of spectral elements meshes can be produced using finite-element mesh generation procedures.

Time integration employs a backwards-time differencing scheme described by Karniadakis et al. (1991), more recently classified as a velocity-correction method by Guermond and Shen (2003). One can select first, second, or third-order time integration, but second order is usually a reasonable compromise, and is the default scheme. Equal-order interpolation is used for velocity and pressure (see Guermond et al.; 2006).

1.2 Implementation

The top level of the code is written in C++, with calls to C and FORTRAN library routines, e.g. BLAS and LAPACK. The original implementation for two-dimensional Cartesian geometries was extended to three dimensions using Fourier expansion functions for spatially-periodic directions in Cartesian and cylindrical spaces. Concurrent execution is supported, using MPI as the basis for interprocessor communications, and the code has been ported to DEC, NEC, Fujitsu, Compaq, SGI, Apple and Linux multiprocessor machines. Basically it ought to work with little trouble on any contemporary UNIX system.

There are various code extensions that are not part of the base distribution. These include dynamic and non-dynamic LES (Blackburn and Schmidt; 2003), simple power-law type non-Newtonian rheologies (Rudman and Blackburn; 2006), scalar transport (Blackburn; 2001, 2002a), buoyancy via the Boussinesq approximation, accelerating frame of reference coupling for aeroelasticity (Blackburn and Henderson; 1996, 1999; Blackburn et al.; 2001; Blackburn; 2003), solution of steady-state flows via Newton-Raphson iteration (Blackburn; 2002b), linear stability analysis (Blackburn; 2002b; Blackburn and Lopez; 2003a,b; Blackburn et al.; 2005; Sherwin and Blackburn; 2005; Elston et al.; 2006; Blackburn and Sherwin; 2007).

1.3 Further reading

The most comprehensive references on spectral methods in general are Gottlieb and Orszag (1977) and Canuto et al. (1988). The first papers by Patera (1984) and Korczak and Patera (1986) provide a good introduction to spectral elements, although some aspects have changed with time and Maday and Patera (1989) is more up-to-date. The use of Fourier expansions to extend the method to three spatial dimensions is discussed by Amon and Patera (1989), Karniadakis (1989) and Karniadakis (1990). The use of spectral element techniques in cylindrical coordinates is dealt with in Blackburn and Sherwin (2004). The book by Funaro (1997) provides useful information and further references. Recent overviews and some applications appear in Karniadakis and Henderson (1998); Henderson (1999). The definitive reference is now the book by Karniadakis and Sherwin (2005), but you will also find the text by Deville, Fischer and Mund (2002) useful for alternative explanations and views.

Chapter 2

Starting out

It is assumed you're using some version of UNIX (this includes Mac OS X), and the current *Semtex* versions assume that your C++ compiler supports the standard libraries. Makefiles assume GNU-make. *SuperMongo* and *Tecplot* would be nice to have but are not essential to get up and running. All major executables have a `-h` command line option which provides a usage prompt.

Application programs/Makefiles can be found in top and upper-level directories:

`elliptic` Solve elliptic (Laplace, Poisson, Helmholtz) problems.
`dns` Solve time-varying Navier–Stokes problems, Cartesian/cylindrical.

The Navier–Stokes solver can also be used to solve unsteady Stokes problems.

2.1 Testing

Unpack the tar file, then run `make test`. This will copy header files to their correct places, compile and place the libraries, make two central utilities (`compare` and `enumerate`), then make the direct numerical simulation solver `dns` and run regression checks on its output for a number of test cases. If all goes well, this process will end with a number of tests reported as `passed`. In that case, go on and compile all the utilities, too (`cd utility; make all`).

2.1.1 Troubleshooting installation problems

If the above process didn't work, there are a number of possible things lacking on your system, e.g.:

1. GNU's `make`: it needs to be in your `path` somewhere under the name `gmake`. On many systems now, `make` *is* `gmake`, in which case you might want to make a soft link from `make` to `gmake`.

2. the BLAS and LAPACK libraries — these may be in `/usr/lib` or `/usr/local/lib`: search for `libblas` and `liblapack`. On many systems, BLAS and LAPACK are vendor-supplied as part of some 'math kernel' library. NB: since quite heavy use is made of BLAS routine `dgemm`, it can be worthwhile finding BLAS versions in which this is well optimised, see § 3.6.

3. a standard C++ compiler, or a FORTRAN-77 compiler. Note that F90, F95 compilers now often supplant F77 compilers, and these can (and should) be substituted if available.

If you have all these things but there are still compilation problems, you have some work to do.

The first place to look is in the file `src/Makefile` which has the master set of compilation flags and directives for various operating systems. You my find your system here, or one that is similar. Even if this is not the case, you should pick up some clues about how to set up for compilation on a new system. As well, check the `README` file in the top directory. Of course, it is possible that the

code is incompatible with some detail of your compilation system or has a bug, but it has had fairly extensive exercise on a number of UNIX systems by now.

If you are having problems, it's usually best to start small and work up. First try to make and install the `veclib` and `femlib` libraries, since they do not use C++ or the linker. Try `make libs` at the top level, then if this is still problematic go to the `veclib` directory, do `make clean; make; make install`. When that works, do the same in the `femlib` directory. Next move on and try a simple C++ compile and link, e.g. in the `utility` directory do `make calc` and try running `calc` (which is a little like the UNIX calculator utility `bc`, but uses *Semtex*'s function parser, and links `libfem.a`, the library produced in `femlib`): try say `1+1`. Next you should try compiling something that links to the BLAS and LAPACK, e.g. `compare`. Once this will compile, everything should.

2.2 Files

Semtex uses a base input file which describes the mesh, boundary conditions. We call this a `session` file and typically it has no root extension. It is written in a format patterned on HTML, which we have called FEML (for Finite Element Markup Language). There are a number of example session files in the mesh directory. Other files have standard extensions:

`session.num` Global node numbers, produced by enumerate utility.
`session.fld` Solution/field file. Binary format by default.
`session.rst` Restart file. Read in to initialize solution if present.
`session.avg` Averaged results. Read back in for continuation (over-written).
`session.his` History point data.
`session.flx` Time series of pressure and viscous forces integrated over the `wall` boundary group.
`session.mdl` Time series of kinetic energies in the Fourier modes (only for 3D execution).
`session.par` Used to define initial particle locations.
`session.trk` Integrated particle locations.

When writing a new session file it is best to run `meshpr` (and/or `meshpr -c`) on it before trying to use it for simulations. `Meshpr` will catch most of the easier-to-make errors. You can also plot up the results using SuperMongo or other utility as a visual check.

2.3 Utilities

Source code for these is found in the `utility` directory. You will need to make most of these by hand (using the supplied `Makefile`, and `make all`). Here is a summary:

`addfield` Add vorticity vector components, divergence, etc., to a field file.
`calc` A simple calculator that calls `femlib`'s function parser. The default functions and `TOKENS` can be seen if you run `calc -h`.
`compare` Generate restart files, compare solutions to a function.
`convert` Convert field file formats (IEEE-big/little, ASCII).
`eneq` Compute terms in the energy transport equation.
`enumerate` Generate global node numbering, with RCM optimization.
`integral` Obtain the 2D integral of fields over the domain area.
`interp` Interpolate a field file onto a (2D) set of points.
`meshpr` Generate 2D mesh locations for plotting or checking.
`noiz` Add a random perturbation to a field file.
`probe` Probe a field file at a set of 2D/3D points. Different interfaces to `probe` are obtained through the names `probeline` and `probeplane`: make these soft links by hand.

`project`	Convert a field file to a different order interpolation.
`rectmesh`	Generate a template session file for a rectangular domain.
`resubmit`	Shell utility for automatic job resubmission.
`rstress`	Postprocess to compute Reyolds stresses from a file of time-averaged variables.
`save`	Shell utility for automatic job resubmssion.
`sem2tec`	Convert field files to Amtec *Tecplot* format.
`transform`	Take Fourier, Legendre, modal basis transform of a field file. Invertible.
`wallmesh`	Extract the mesh nodes corresponding to surfaces with the `wall` group.

Chapter 3

Examples and hints

We will run through some examples to illustrate input files, utility routines, and the use of the solvers.

3.1 2D Taylor flow

Taylor flow is an analytical solution to the Navier–Stokes equations. In the x–y plane the solution is

$$u = -\cos(\pi x)\sin(\pi y)\exp(-2\pi^2\nu t), \tag{3.1}$$
$$v = +\sin(\pi x)\cos(\pi y)\exp(-2\pi^2\nu t), \tag{3.2}$$
$$p = -(\cos(2\pi x)+\cos(2\pi y))\exp(-4\pi^2\nu t)/4. \tag{3.3}$$

The solution is doubly periodic in space, with periodic length 2. As usual for Navier–Stokes solutions, the pressure can only be specified up to an arbitrary constant. An interesting feature of this solution is that the nonlinear and pressure gradient terms balance one another, leaving a diffusive decay of the initial condition — this property is occasionally useful for checking codes.

3.1.1 Session file

Below is the complete input or *session* file we will use; it has four elements, each of the same size, with 11 nodes along each edge. We will call this session file `taylor2` in the following.

```
#############################################################################
# 2D Taylor flow in the x--y plane has the exact solution
#
#       u = -cos(PI*x)*sin(PI*y)*exp(-2.0*PI*PI*KINVIS*t)
#       v =  sin(PI*x)*cos(PI*y)*exp(-2.0*PI*PI*KINVIS*t)
#       w =  0
#       p = -0.25*(cos(2.0*PI*x)+cos(2.0*PI*y))*exp(-4.0*PI*PI*KINVIS*t)
#
# Use periodic boundaries (no BCs).

<USER>
        u = -cos(PI*x)*sin(PI*y)*exp(-2.0*PI*PI*KINVIS*t)
        v =  sin(PI*x)*cos(PI*y)*exp(-2.0*PI*PI*KINVIS*t)
        p = -0.25*(cos(TWOPI*x)+cos(TWOPI*y))*exp(-4.0*PI*PI*KINVIS*t)
</USER>

<FIELDS>
        u v p
</FIELDS>

<TOKENS>
```

```
        N_TIME  = 2
        N_P     = 11
        N_STEP  = 20
        D_T     = 0.02
        Re      = 100.0
        KINVIS  = 1.0/Re
        TOL_REL = 1e-12
</TOKENS>

<NODES NUMBER=9>
        1       0.0       0.0       0.0
        2       1.0       0.0       0.0
        3       2.0       0.0       0.0
        4       0.0       1.0       0.0
        5       1.0       1.0       0.0
        6       2.0       1.0       0.0
        7       0.0       2.0       0.0
        8       1.0       2.0       0.0
        9       2.0       2.0       0.0
</NODES>

<ELEMENTS NUMBER=4>
        1 <Q> 1 2 5 4 </Q>
        2 <Q> 2 3 6 5 </Q>
        3 <Q> 4 5 8 7 </Q>
        4 <Q> 5 6 9 8 </Q>
</ELEMENTS>

<SURFACES NUMBER=4>
        1       1       1       <P>     3       3       </P>
        2       2       1       <P>     4       3       </P>
        3       2       2       <P>     1       4       </P>
        4       4       2       <P>     3       4       </P>
</SURFACES>
```

The first section of the file in this case contains comments; a line anywhere in the session file which starts with a # is considered to be a comment. Following that are a number of sections which are opened and closed with matching keywords in HTML style (e.g. <USER>–<\USER>). Keywords are not case sensitive. The complete list of keywords is: TOKENS, FIELDS, GROUPS, BCS, NODES, ELEMENTS, SURFACES, CURVES and USER. Depending on the problem being solved, some sections may not be needed, but the minimal set is: FIELDS, NODES, ELEMENTS and SURFACES. Anywhere there is likely to be a long list of inputs within the sections, the NUMBER of inputs is also required; this currently applies to GROUPS, BCS, NODES, ELEMENTS, SURFACES and CURVES. In each of these cases the numeric tag appears first for each input, which is free-format. The order in which the sections appear in the session file is irrelevant.

The USER section is ignored by the solvers, and is used instead by utilities — in this case it will be used by the compare utility both to generate the initial condition or *restart* file and to check the computed solution. This section declares the variables corresponding to the solution fields with the corresponding analytical solutions. The variables x, y, z and t can be used to represent the three spatial coordinates and time. Note that some constants such as PI and TWOPI are predefined, while others, like KINVIS, are set in the TOKENS section. Note also the use of predefined functions, accessed through an inbuilt function parser[1].

The FIELDS section declares the one-character names of solution fields. The names are significant: u, v and w are the three velocity components (we only use u and v here for a 2D solution and the w

[1] The built-in functions and predefined constants can be found by running calc -h.

component is always the direction of Fourier expansions), p is the pressure field. The field name c is also recognized as a scalar field for certain solvers, e.g. the elliptic solver.

In the TOKENS section, second-order accurate time integration is selected (N_TIME = 2) and the number of Lagrange knot points along the side of each element is set to 11 (N_P = 11). The code will integrate for 20 timesteps (N_STEP = 20) with a timestep of 0.02 (D_T = 0.002). The kinematic viscosity is set as the inverse of the Reynolds number (100): note the use of the function parser here. Finally the relative tolerance used as a stopping test in the PCG iteration used to solve the viscous substep on the first timestep is set as 1.0×10^{-12}.

The shape of the mesh is defined by the NODES and ELEMENTS sections. Here there are four elements, each obtained by connecting the corner nodes in a counterclockwise traverse. The x, y and z locations of the nodes are given, and the four numbers given for the nodes of each element are indices within the list of nodes.

In the final section (SURFACES), we describe how the edges of elements which define the boundary of the solution domain are dealt with. In this example, the solution domain is periodic and there are no boundary conditions to be applied, so the SURFACES section describes only periodic (P) connections between elements. For example, on the first line, side 1 of element 1 is declared to be periodic with side 3 of element 3 — side 1 runs between the first and second nodes, while side 3 runs between the third and fourth.

3.1.2 Running the codes

Assume we're in the dns directory of the distribution, that the enumerate, compare, meshpr and sem2tec utilities have been compiled, as well as the dns simulation code.

```
karman[16] cp ../mesh/taylor2 .
```

First we'll examine the mesh, using SuperMongo macros.

```
karman[17] meshpr taylor2 > taylor2.msh
karman[18] sm
Hello Hugh, please give me a command
: meshplot taylor2.msh 1
Read lines 1 to 1 from taylor2.msh
Read lines 2 to 485 from taylor2.msh
: meshnum
: meshbox
: quit
```

You should have seen a plot like that in figure 3.1. (Note: while you are building up the mesh parts of a session file, you can use meshpr -c to suppress some of the checking for matching element edges and curved boundaries that meshpr does by default.)

Next we will generate the global numbering schemes for the solution using enumerate to produce taylor2.num. The solution code would run enumerate automatically to generate taylor2.num if it were not present, but we will run it 'by hand' to highlight its existence and illustrate its use.

```
karman[19] enumerate taylor2 > taylor2.num
karman[20] head -20 taylor2.num
# FIELDS         :  uvp
# ----------------  ----------
# 1 NUMBER SETS  :         uvp
# NEL            :           4
# NP_MAX         :          11
# NEXT_MAX       :          40
# NINT_MAX       :          81
# NTOTAL         :         484
# NBOUNDARY      :         160
# NGLOBAL        :          76
```

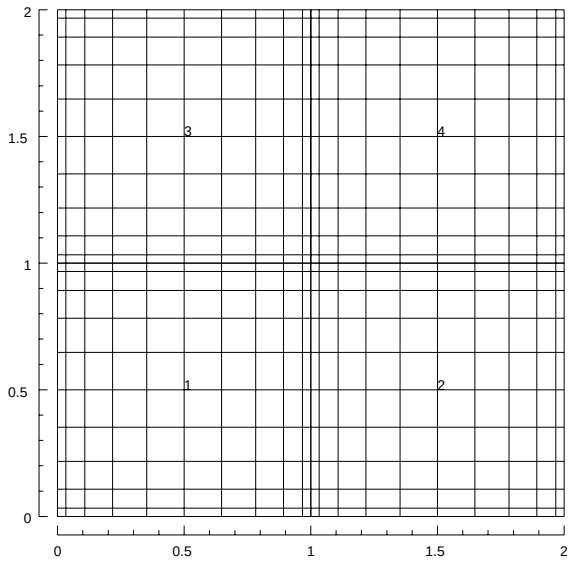


Figure 3.1: The mesh corresponding to the `taylor2` session file.

```
# NSOLVE         :          76
# OPTIMIZATION   :           1
# BANDWIDTH      :          67
# ----------------  ----------
# elmt  side offst  bmap  mask
     1     1     0    20     0
     1     1     1    17     0
     1     1     2    16     0
     1     1     3    15     0
     1     1     4    14     0
```

The `compare` utility is used to generate a file of initial conditions. This *restart* file contains binary data, but we'll have a look at the start of it by converting it to ASCII format. Also, the header of these files is always in ASCII format, and so can be examined directly using the Unix `head` command.

```
karman[21] compare taylor2 > taylor2.rst
karman[22] convert taylor2.rst | head -20
taylor2                  Session
Wed Aug 13 21:39:47 1997  Created
11   11   1    4          Nr, Ns, Nz, Elements
0                        Step
0                        Time
0.02                     Time step
0.01                     Kinvis
1                        Beta
uvp                      Fields written
ASCII                    Format
     0.000000000      0.000000000    -0.5000000000
     0.000000000     0.1034847104    -0.4946454574
     0.000000000     0.3321033052    -0.4448536974
     0.000000000     0.6310660897    -0.3008777952
     0.000000000     0.8940117093    -0.1003715318
     0.000000000      1.000000000      0.000000000
     0.000000000     0.8940117093    -0.1003715318
     0.000000000     0.6310660897    -0.3008777952
```

```
     0.000000000     0.3321033052    -0.4448536974
     0.000000000     0.1034847104    -0.4946454574
```

Then the `dns` solver is run to generate a solution or *field* file, `taylor2.fld`. This has the same format as the restart file.

```
karman[23] dns taylor2
-- Restarting from file:  taylor2.rst
   Start time       : 0
   Time step        : 0.02
   Number of steps  : 20
   End time         : 0.4
   Integration order: 2
-- Building matrices for Fields "uvp"   [*]
-- Building matrices for Fields "uvp"   [.]
-- Building matrices for Fields "uvp"   [*]
Step: 1  Time: 0.02
Step: 2  Time: 0.04
Step: 3  Time: 0.06
Step: 4  Time: 0.08
Step: 5  Time: 0.1
Step: 6  Time: 0.12
Step: 7  Time: 0.14
Step: 8  Time: 0.16
Step: 9  Time: 0.18
Step: 10  Time: 0.2
Step: 11  Time: 0.22
Step: 12  Time: 0.24
Step: 13  Time: 0.26
Step: 14  Time: 0.28
Step: 15  Time: 0.3
Step: 16  Time: 0.32
Step: 17  Time: 0.34
Step: 18  Time: 0.36
Step: 19  Time: 0.38
Step: 20  Time: 0.4
```

We can use `compare` to examine how close the solution is to the analytical solution. The output of `compare` in this case is a field file which contains the difference: since we're only interested in seeing error norms here, we'll discard this field file.

```
karman[24] compare taylor2 taylor2.fld > /dev/null
Field 'u': norm_inf: 1.13019e-05
Field 'v': norm_inf: 1.13019e-05
Field 'p': norm_inf: 0.422391
```

The velocity error norms are small, as expected, but the pressure norm will always be arbitrary, corresponding to the fact that the pressure can only be specified to within an arbitrary constant.

Finally we will use `sem2tec` to generate a *Tecplot* input file. The *Tecplot* utility `preplot` must also be in your `path`.

```
karman[36] sem2tec -m taylor2.msh taylor2.fld
```

This produces `taylor2.plt` which can be used as input to `tecplot`. The plot in figure 3.2 was generated using `tecplot` and shows pressure contours and velocity vectors. Notice that by default `sem2tec` interpolates the results from the Gauss–Lobatto–Legendre grid (seen in figure 3.1) used in the computation to a uniformly-spaced grid of the same order.

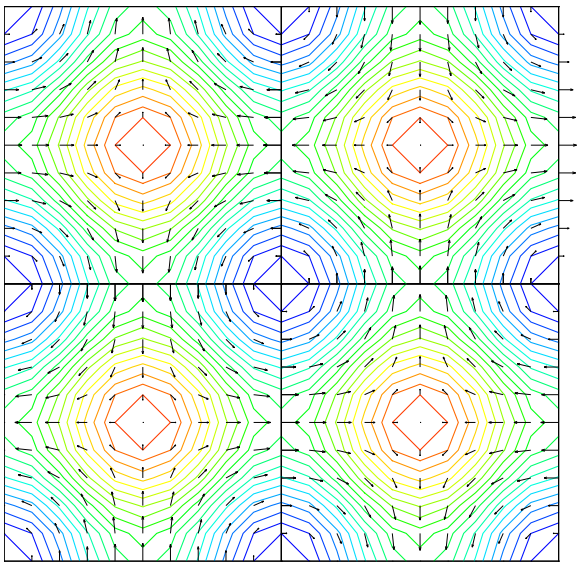


Figure 3.2: Solution to the `taylor2` problem, visualized using *Tecplot*.

3.2 2D Laplace problem

In this section we illustrate the use of the elliptic solver for a 2D Laplace problem, $\nabla^2 c = 0$. In this case the function

$$c(x, y) = \sin(x) \exp(-y) \tag{3.4}$$

satisfies Laplace's equation and is used to set the boundary conditions. This example illustrates the methods used to set BCs and also to generate curved element boundaries. Also we will demonstrate the selection of the PCG solver. We will call the session file `laplace6`.

The elliptic solver can also be used to solve Poisson and Helmholtz problems in 2D and 3D Cartesian and cylindrical coordinate systems. Apart from this use it provides a means to test new formulations of elliptic solution routines used also in the Navier–Stokes type solvers.

```
################################################################################
# Laplace problem on unit square, BC c(x, y) = sin(x)*exp(-y)
# is also the analytical solution.  Use essential (Dirichlet) BC
# on upper, curved edge, with natural (Neumann) BCs elsewhere.

<FIELDS>
        c
</FIELDS>

<USER>
        c =  sin(x)*exp(-y)
</USER>

<TOKENS>
        N_P      = 11
        TOL_REL  = 1e-12
        STEP_MAX = 1000
</TOKENS>

<GROUPS NUMBER=4>
        1       d       value
```

```
        2       a       slope
        3       b       slope
        4       c       slope
</GROUPS>

<BCS NUMBER=4>
        1       d       1
                        <D>     c =  sin(x)*exp(-y)     </D>
        2       a       1
                        <N>     c = -cos(x)*exp(-y)     </N>
        3       b       1
                        <N>     c =  cos(x)*exp(-y)     </N>
        4       c       1
                        <N>     c =  sin(x)*exp(-y)     </N>
</BCS>

<NODES NUMBER=9>
        1       0.0     0.0     0.0
        2       0.5     0.0     0.0
        3       1.0     0.0     0.0
        4       0.0     0.5     0.0
        5       0.5     0.5     0.0
        6       1.0     0.5     0.0
        7       0.0     1.0     0.0
        8       0.5     1.0     0.0
        9       1.0     1.0     0.0
</NODES>

<ELEMENTS NUMBER=4>
        1       <Q>     1 2 5 4         </Q>
        2       <Q>     2 3 6 5         </Q>
        3       <Q>     4 5 8 7         </Q>
        4       <Q>     5 6 9 8         </Q>
</ELEMENTS>

<SURFACES NUMBER=8>
        1       1       1       <B>     c       </B>
        2       2       1       <B>     c       </B>
        3       2       2       <B>     b       </B>
        4       4       2       <B>     b       </B>
        5       4       3       <B>     d       </B>
        6       3       3       <B>     d       </B>
        7       3       4       <B>     a       </B>
        8       1       4       <B>     a       </B>
</SURFACES>

<CURVES NUMBER=1>
        1       4       3       <ARC>   1.0     </ARC>
</CURVES>
```

3.2.1 Curved element edges

The mesh is somewhat similar to that for the Taylor flow example, the only difference being in the use of a curved edge for the 3rd edge of element 4, as specified in the `CURVES` section. Both `ARC` and `SPLINE` type curves are currently implemented. For the `ARC` type, the parameter supplies the radius of the curve: a positive value implies that the curve makes the element convex on that side, while a negative value implies a concave side. Note that where elements mate along a curved side,

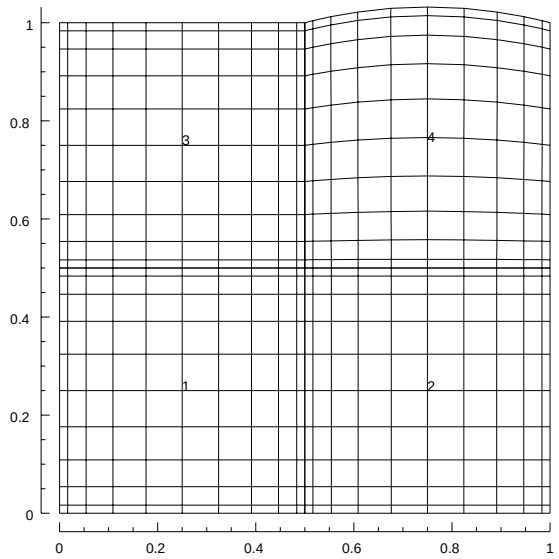


Figure 3.3: The mesh corresponding to the `laplace6` session file.

the curve must be defined twice, once for each element, and with radii of different signs on each side. The mesh for this problem can be seen in figure 3.3.

For the `SPLINE` type, the parameter supplies the name of an ASCII file which contains a list of (x, y) coordinate pairs (white-space delimited). Naturally, the list of points should be in arc-length order. A single file can be used to supply the curved edges for a set of element edges. The vertices of the relevant elements do not have to lie exactly on the splined curve — if they do not, those vertices get shifted to the intersection of the projection of the straight line joining the original vertex position and its neighbouring "curve-normal' vertex, and the cubic spline joining the points in the file. On the other hand, it is good practise to ensure that the declared vertex locations lie close to the spline, and to check the mesh that is produced: use `meshpr` to do this.

3.2.2 Boundary conditions

The other new sections introduced in the session file for this example (`GROUPS`, `BCS`) are used to impose boundary conditions on the problem. The `GROUPS` section associates a character group tag (e.g. `d`) with a string (e.g. `value`), but note that different groups can be associated with the same string[2]. Groups `a`, `b` and `c` will be used to set natural (i.e. slope or Neumann), boundary conditions, while group `a` will be used to impose an essential or Dirichlet condition.

The `BCS` section is used to define the boundary conditions which will be applied for each group. For each group, after the numeric tag (ignored) appears the character for that group, then the number of BCs that will be applied: this corresponds to the number of fields in the problem, in this case 1 (`c`). BCs can be only of Dirichlet or Neumann type — mixed BCs are not implemented yet. So in this case we will declare the BC types to be `D` (Dirichlet) for group `d` and `N` (Neumann) for groups `a`, `b` and `c`. On Neumann boundaries, the value which must be supplied is the slope of the solution along the outward normal to the solution domain. Note the fact that the BCs can be set using the function parser, using the built-in functions and variables, also any symbols defined in the `TOKENS` section. The BCs can be functions of time, `t`, but in time-varying problems the BCs are only set at the initial time, and not subsequently re-parsed.

The BC groups are associated with element edges in the `SURFACES` section, in a similar way to

[2]This allows actions to be taken over a set of BCs which share the same string.

the use of periodic boundaries for the `taylor2` problem, although the edges are set to be `B` (BC) rather than `P` (periodic).

Periodic, Dirichlet and Neumann boundary conditions can be arbitrarily mixed in a problem. Dirichlet conditions over-ride Neumann ones where they meet (say at the corner node of an element).

3.2.3 Running the codes

We will run the solver and compare the computed solution to the analytical solution. We will select the iterative (PCG) solver using the `-i` command-line option to `elliptic`, then check the result using `compare`.

```
karman[287] elliptic -i laplace6
-- Initializing solution with zero IC
   Start time       : 0
   Time step        : 0.01
   Number of steps  : 1
   End time         : 0.01
   Integration order: 2
karman[288] compare laplace6 laplace6.fld > /dev/null
Field 'c': norm_inf: 1.37112e-08
```

Next try the direct solver (default):

```
karman[288] compare laplace6 laplace6.fld > /dev/null
Field 'c': norm_inf: 1.37112e-08
karman[289] elliptic laplace6
-- Initializing solution with zero IC
   Start time       : 0
   Time step        : 0.01
   Number of steps  : 1
   End time         : 0.01
   Integration order: 2
-- Building matrices for Fields "c"     [*]
karman[290] compare laplace6 laplace6.fld > /dev/null
Field 'c': norm_inf: 3.10862e-15
```

In this case the direct solver is more accurate, but this could be changed by decreasing `TOL_REL` in the `TOKENS` section (and further increasing `STEP_MAX`, which has a default value of 500).

3.3 3D Kovasznay flow

Here we will solve another viscous flow for which an analytical solution exists, the Kovasznay flow (we will call the session file `kovas3`). In the x–y plane, this flow is

$$u = 1 - \exp(\lambda x)\cos(2\pi y) \tag{3.5}$$

$$v = \lambda/(2\pi)\exp(\lambda x)\sin(2\pi y) \tag{3.6}$$

$$w = 0 \tag{3.7}$$

$$p = (1 - \exp(\lambda x))/2 \tag{3.8}$$

where $\lambda = Re/2 - (0.25Re^2 + 4\pi^2)^{1/2}$.

Although the solution has only two velocity components, we will set up and solve the problem in three dimensions, with a periodic length in the z direction of 1.0 and 8 z planes of data. The length in the z direction is set within the code by the variable `BETA` where $\beta = 2\pi/L_z$. The default value of `BETA` is 1, so we reset this in the `TOKENS` section using the function parser. The exact velocity boundary conditions are supplied on at the left and right edges of the domain, and periodic boundaries are used on the upper and lower edges (the domain has $-0.5 \leq y \leq 0.5$). Since the flow evolves to a steady state, first order timestepping is employed (`N_TIME = 1`).

```
################################################################################
# Kovasznay flow in the x--y plane has the exact solution
#
#       u = 1 - exp(lambda*x)*cos(2*PI*y)
#       v = lambda/(2*PI)*exp(lambda*x)*sin(2*PI*y)
#       w = 0
#       p = (1 - exp(lambda*x))/2
#
# where lambda = Re/2 - sqrt(0.25*Re*Re + 4*PI*PI).
#
# This 3D version uses symmetry planes on the upper and lower boundaries
# with flow in the x-y plane.
#
# Solution accuracy is independent of N_Z since all flow is in the x--y plane.

<USER>
        u = 1.0-exp(LAMBDA*x)*cos(TWOPI*y)
        v = LAMBDA/(TWOPI)*exp(LAMBDA*x)*sin(TWOPI*y)
        w = 0.0
        p = 0.5*(1.0-exp(LAMBDA*x))
</USER>

<FIELDS>
        u v w p
</FIELDS>

<TOKENS>
        N_Z    = 8
        N_TIME = 1
        N_P    = 8
        N_STEP = 500
        D_T    = 0.008
        Re     = 40.0
        KINVIS = 1.0/Re
        LAMBDA = Re/2.0-sqrt(0.25*Re*Re+4.0*PI*PI)
        Lz     = 1.0
        BETA   = TWOPI/Lz
</TOKENS>

<GROUPS NUMBER=1>
        1       v       velocity
</GROUPS>

<BCS NUMBER=1>
        1       v       4
                        <D> u = 1-exp(LAMBDA*x)*cos(2*PI*y)             </D>
                        <D> v = LAMBDA/(2*PI)*exp(LAMBDA*x)*sin(2*PI*y) </D>
                        <D> w = 0.0                                     </D>
                        <H> p                                           </H>
</BCS>

<NODES NUMBER=9>
        1      -0.5    -0.5     0.0
        2       0      -0.5     0.0
        3       1      -0.5     0.0
        4      -0.5     0       0.0
        5       0       0       0.0
        6       1       0       0.0
```

```
        7      -0.5     0.5     0.0
        8       0       0.5     0.0
        9       1       0.5     0.0
</NODES>

<ELEMENTS NUMBER=4>
        1 <Q> 1 2 5 4 </Q>
        2 <Q> 2 3 6 5 </Q>
        3 <Q> 4 5 8 7 </Q>
        4 <Q> 5 6 9 8 </Q>
</ELEMENTS>

<SURFACES NUMBER=6>
        1       1       1       <P>     3       3       </P>
        2       2       1       <P>     4       3       </P>
        3       2       2       <B>     v               </B>
        4       4       2       <B>     v               </B>
        5       3       4       <B>     v               </B>
        6       1       4       <B>     v               </B>
</SURFACES>
```

3.3.1 'High-order' pressure boundary condition

Note that there is only one boundary group, and four boundary conditions must be set, corresponding to the four fields `u`, `v`, `w` and `p`. A new feature is a pressure BC of type `H`, which is an internally-computed Neumann boundary condition, (a High-order pressure BC) as described in Karniadakis et al. (1991). This is the kind of pressure BC that is supplied at all places except on outflow boundaries. The pressure BC is computed internally, so no value is required (if given, it will be ignored).

3.3.2 Running the codes

After running `dns`, we confirm there is only a single dump in the field file `kovas3.fld`, then run compare in order to examine the error norms for the solution. Following that we prepare input for *Tecplot*, projecting the interpolation to a 20×20 grid in each element. A view of the result can be seen in figure 3.4.

```
karman[25] convert kovas3.fld | grep -i session
kovas3                    Session
karman[26] compare kovas3 kovas3.fld > /dev/null
Field 'u': norm_inf: 5.70744e-05
Field 'v': norm_inf: 3.04095e-05
Field 'w': norm_inf: 0
Field 'p': norm_inf: 0.928545
karman[27] meshpr kovas3 | sem2tec -n20 kovas3.fld
```

3.4 Vortex breakdown — a cylindrical-coordinate problem

Here we will examine a problem which uses the cylindrical coordinate option of `dns`. The physical situation is a cylindrical cavity, $H/R = 2.5$ with the flow driven by a spinning lid at one end. At the Reynolds number we'll use, $Re = \Omega R^2/\nu = 2119$, a vortex breakdown is known to occur. The flow in this case is invariant in the azimuthal direction, but has three velocity components (it is 2D/3C). In the cylindrical code, the order of spatial directions and velocity components is z, r, θ.

Note that for a full circle in the azimuthal direction, `BETA = 1.0`, (which is the default value). In fact, the value would not be used in the present solution, since all derivatives in the azimuthal direction are implicitly zero when `N_Z=1`. (But see § 3.4.1 below.)

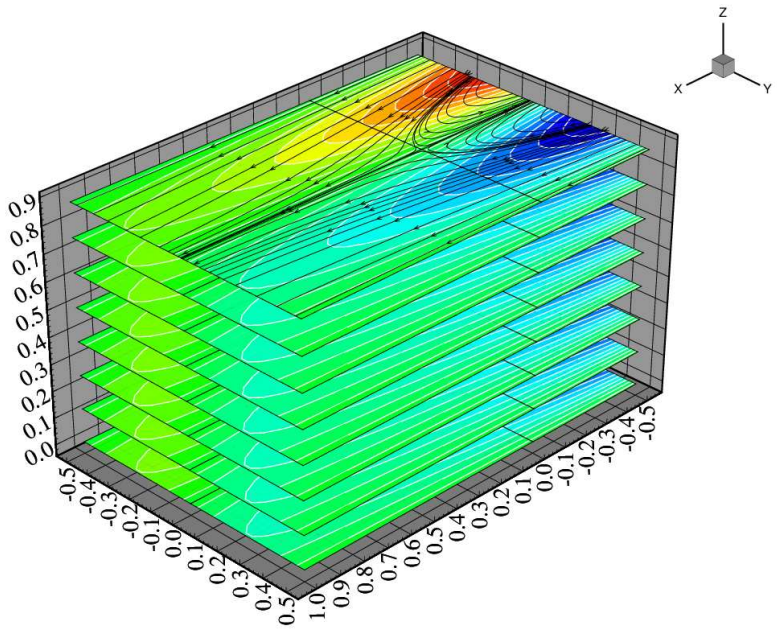


Figure 3.4: Solution to the `kovas3` problem, visualized using *Tecplot*. The plot shows contours of v velocity component and streamlines.

```
################################################################################
# 15 element driven cavity flow.

<FIELDS>
        u v w p
</FIELDS>

<TOKENS>
        CYLINDRICAL = 1
        N_Z         = 1
        BETA        = 1.0
        N_TIME      = 2
        N_P         = 11
        N_STEP      = 100000
        D_T         = 0.01
Re          = 2119
        KINVIS      = 1/Re
        OMEGA       = 1.0
        TOL_REL     = 1e-12
</TOKENS>

<GROUPS NUMBER=3>
        1       v       velocity
        2       w       wall
        3       a       axis
</GROUPS>

<BCS NUMBER=3>
        1       v       4
                        <D>     u = 0           </D>
                        <D>     v = 0           </D>
                        <D>     w = OMEGA*y     </D>
```

```
                        <H>     p               </H>
        2       w       4
                        <D>     u = 0           </D>
                        <D>     v = 0           </D>
                        <D>     w = 0           </D>
                        <H>     p               </H>
        3       a       4
                        <A>     u               </A>
                        <A>     v               </A>
                        <A>     w               </A>
                        <A>     p               </A>
</BCS>

<NODES NUMBER=24>
        1       0       0       0
        2       0.4     0       0
        3       0.8     0       0
        4       1.5     0       0
        5       2.4     0       0
        6       2.5     0       0
        7       0       0.15    0
        8       0.4     0.15    0
        9       0.8     0.15    0
        10      1.5     0.15    0
        11      2.4     0.15    0
        12      2.5     0.15    0
        13      0       0.75    0
        14      0.4     0.75    0
        15      0.8     0.75    0
        16      1.5     0.818   0
        17      2.4     0.9     0
        18      2.5     0.9     0
        19      0       1       0
        20      0.4     1       0
        21      0.8     1       0
        22      1.5     1       0
        23      2.4     1       0
        24      2.5     1       0
</NODES>

<ELEMENTS NUMBER=15>
        1       <Q>      1  2  8  7     </Q>
        2       <Q>      2  3  9  8     </Q>
        3       <Q>      3  4 10  9     </Q>
        4       <Q>      4  5 11 10     </Q>
        5       <Q>      5  6 12 11     </Q>
        6       <Q>      7  8 14 13     </Q>
        7       <Q>      8  9 15 14     </Q>
        8       <Q>      9 10 16 15     </Q>
        9       <Q>     10 11 17 16     </Q>
        10      <Q>     11 12 18 17     </Q>
        11      <Q>     13 14 20 19     </Q>
        12      <Q>     14 15 21 20     </Q>
        13      <Q>     15 16 22 21     </Q>
        14      <Q>     16 17 23 22     </Q>
        15      <Q>     17 18 24 23     </Q>
</ELEMENTS>
```

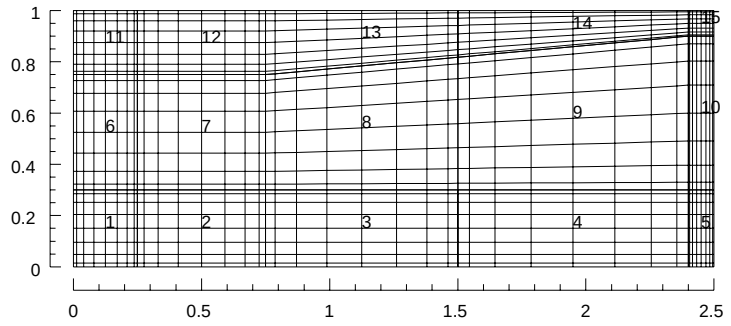


Figure 3.5: Mesh for the vortex breakdown problem. The spinning lid is at right.

```
<SURFACES NUMBER=16>
        1       1       1       <B>     a       </B>
        2       2       1       <B>     a       </B>
        3       3       1       <B>     a       </B>
        4       4       1       <B>     a       </B>
        5       5       1       <B>     a       </B>
        6       5       2       <B>     v       </B>
        7       10      2       <B>     v       </B>
        8       15      2       <B>     v       </B>
        9       15      3       <B>     w       </B>
        10      14      3       <B>     w       </B>
        11      13      3       <B>     w       </B>
        12      12      3       <B>     w       </B>
        13      11      3       <B>     w       </B>
        14      11      4       <B>     w       </B>
        15      6       4       <B>     w       </B>
        16      1       4       <B>     w       </B>
</SURFACES>
```

The mesh for the problem is shown in figure 3.5, and the velocity field is shown compared to an experimental streakline flow visualisation on the front cover of this document.

3.4.1 BCs for cylindrical coordinates

A new feature here is the use of BCs of type `A` on the axis of the flow. Internally, the code sets the BC there either as zero essential or zero natural, depending on the physical variable and the Fourier mode. Owing to the coupling scheme used in the code (Blackburn and Sherwin; 2004), the boundary conditions for the radial and azimuthal velocities `v` and `w` must be of the same type within each group. A further restriction is that the group to which the axis belongs must have name `axis`.

Finally, say you wish to solve a cylindrical-coordinate problem where there is an n-fold azimuthal symmetry (say $n = 3$). In that case, it is much cheaper to solve with `BETA=3`, and use $1/3$ the number of azimuthal planes that would be required for `BETA=1`. However, if the axis is included in the domain, you should also set `K_FUND=3` — this ensures that the correct axial boundary conditions (which are Fourier-mode-number dependent) get selected by the code. Usually I set say `K_FUND=3` and `BETA=K_FUND`.

3.5 Fixing problems

You are liable to come up against a few generic problems when making and running your own cases. Here we will restrict discussion to Navier–Stokes problems and `dns`. The best diagnostic of trouble is

the divergence of the solution. The code will output an estimate of the CFL-timestep every IO_CFL timesteps (default 50), along with the average divergence of the solution (in the operator-splitting used, incompressibility is only ensured in the spatial-convergence limit). Unfortunately the CFL estimate is presently rather unreliable, but the divergence energy provides an excellent diagnostic of trouble! If velocity and length scales are of order unity, the reported divergence energy should be much less than unity; if the divergence is large then either the solution is blowing up[3], or the spatial resolution is inadequate, or both.

By far the most common problem is that the solution will have a CFL-type instability brought about by using too large a time-step; this instability is unavoidably associated with using explicit time integration for the advection terms in the Navier–Stokes equations. This problem is easily enough fixed: try reducing D_T and increasing N_STEP to maintain the same integration interval. Obviously you will typically want D_T as large as possible, so if the problem runs stably, increase the timestep as much as is reasonable. If the velocity and timescales are of order unity, then the maximum timestep would typically be of order two orders of magnitude smaller (0.01). Note that CFL-stability will decrease with increasing time-integration order (N_TIME).

If the solution persists in blowing up when the timestep is reduced, the next most common cause is that there is inflow across an outflow boundary (in which case the problem is ill-posed, however, in practice *some* inflow across an outflow boundary over restricted times may be present without causing difficulty). To check if this is the cause, you could put some history points near the outflow (see § 4.7), but the best method is to run the solution up to a time when divergence starts to increase markedly, then use *Tecplot* or some other postprocessor to examine the solution near the outflow. Fixing the problem will generally require the mesh to be altered: sometimes the mesh is badly structured near the outflow (e.g. element sizes have been varied too rapidly); sometimes the problem can be overcome by extending the domain downstream; sometimes the domain needs to be reshaped (e.g. by contracting it in the cross-flow direction) so that there will be no outflow over the inflow boundary. But unfortunately, sometimes one will conclude that no matter how the problem is tackled, there will be inflow somewhere over the outflow boundary: try a different problem.

Any time you change the element polynomial order by changing N_P, or alter the structure of the boundary conditions, you should remake the numbering file session.num. It is especially important to remember this if you have changed the structure of the boundary conditions without changing element order, as no warning will be triggered; however, in this case you may no longer be actually applying the boundary conditions you have set in the session file because the mask values in session.num (which are set to 1 along a Dirichlet-type boundary) may no longer match what is implied by session.

3.6 Speed

Semtex relies heavily on the BLAS, so it can be worth seeking fast implementations. Especially, performance of matrix–matrix multiplication routine dgemm is critical. Consistently of late, Kazushige Goto's implementation of the BLAS gives best performance and is worth seeking out, although I understand his dgemm has also now been licenced to various vendors and is what you'll get if you link their versions of the BLAS.

As Reynolds numbers increase (i.e. KINVIS decreases), the viscous Helmholtz matrices in the operator splitting become more diagonally dominant and better conditioned. In this case, you may find that iterative solution of the viscous step (obtained by setting ITERATIVE=1 or running dns -i) is actually faster than the direction solution that is obtained by default. This is nice because additionally, less memory is required. It is generally worth checking this if you plan an extended series of runs, and Reynolds numbers are large.

[3] One wag suggested the name *Semtex* was associated with this property of the solutions.

Chapter 4

Extra controls

This chapter describes some additional features that are implemented within the Navier–Stokes solver `dns` to control execution and output.

4.1 Default values of flags and internal variables

There are two simple ways to establish the default values of all the internal flags and variables used by *Semtex*. The first is via the `calc` utility: run `calc -h` and check the output (this will also show you all the functions available to the parser for calculating `TOKEN` variables, initial and boundary conditions). The second is to examine the file `femlib/defaults.h`.

4.2 Checkpointing

By default, intermediate solutions are appended to `session.fld` every `IO_FLD` steps (default value `IO_FLD = 500`). For lengthy or 3D simulations, `session.fld` can quickly grow to an unreasonable size. Controlling this by setting `IO_FLD = N_STEPS` can be risky, as the simulation could become unstable or be terminated before ending as expected. A better approach (unless the intermediate fields are desired for postprocessing) is to use checkpointing. With `CHKPOINT = 1`, intermediate field dumps are written to `session.chk`, rotating to `session.chk.bak` every `IO_FLD` steps, until the end of the simulation, when `session.fld` is written. A command line option (`dns -chk`) can also be used to select checkpointing, but this is over-ridden by whatever is set in the `session` file.

4.3 Iterative solution

Two matrix solution methods are implemented for Helmholtz problems associated with the pressure and viscous substeps of the time splitting. By default, direct Schur-complement solutions are used. The associated global matrices can consume quite large amounts of memory, typically much more than is required for storage of the associated field variable. Iterative (PCG) solution can also be selected, and this has the advantage that since it is matrix-free, no global matrices are required, however, solution may be slower than for the direct solver (depending on the condition number of the global matrix problem).

The token that controls the selection of solution method is `ITERATIVE`. For `dns`, PCG solution can be selected for the viscous substep of the solution (`ITERATIVE = 1`), or for both viscous and pressure substeps (`ITERATIVE = 2`). The same functions can be selected by command-line options (`dns -i` $\equiv$ `ITERATIVE = 1`, while `dns -ii` $\equiv$ `ITERATIVE = 2`), but note that these options are overridden by whatever tokens are set in the `session` file (the default value is `ITERATIVE = 0`).

Iterative solution is most useful for the viscous substep, particularly when the Reynolds number is high, since this decreases the condition number of the associated global matrices. In fact, iterative

solutions for the viscous substep can execute faster than direct solutions at high Reynolds number, although this is platform dependent. Since the global pressure matrices have higher condition numbers, particularly for low Fourier modes, iterative solution for the pressure substep is only of any practical use when memory at a premium.

4.4 Wall fluxes

A file called `session.flx` is used to store the integral over the `wall` group boundaries of viscous and pressure stresses (i.e. lift and drag forces). Output is done every `IO_HIS` steps. For each direction (x, y, z), the outputs are in turn the pressure, viscous, and total force per unit length. In 2D the z-components are always zero, while in 3D the z-component pressure force is always zero, owing to the fact that the geometry is invariant in that direction. For cylindrical geometries, the output values are forces per radian (in the x and y directions) and torque per radian (in the z direction) rather than forces per unit length.

4.5 Wall tractions

If the token `IO_WSS` is set to a non-zero value then the normal and the single (2D) or two (3D) components of tangential boundary traction are computed on the `wall` group, and output every `IO_WSS` steps in the file `session.wss`. This is a binary file with structure similar to a field dump. The utility `wallmesh` is used to extract the corresponding mesh points along the walls.

4.6 Modal energies

For 3D simulations (`N_Z > 2`), a file of modal energies, `session.mdl`, is produced. This provides valuable diagnostic information for turbulent flow simulations. For each active Fourier mode k in the simulation, the value output every `IO_HIS` steps is

$$E_k = \frac{1}{2A} \int_\Omega \hat{\boldsymbol{u}}_k^* \cdot \hat{\boldsymbol{u}}_k \, \mathrm{d}\Omega,$$

where A is the area of the 2D domain Ω. (In cylindrical coordinate problems, the integrand is multiplied by radius.) Each line of the file contains the time t, mode number k and E_k.

4.7 History points

History points are used to record solution variables at fixed spatial locations as the simulation proceeds. The locations need not correspond to grid points, as data are interpolated onto the given spatial locations using the elemental basis functions. Locations of history points are declared in the `session` file as follows:

```
<HISTORY NUMBER=1>
#       tag     x       y       z
        1       0       0       0
</HISTORY>
```

A file called `session.his` is produced as output. Each line of the file contains the step number, the time, the history point tag number, followed by values for each of the solution variables. The step interval at which history point information is dumped to file is controlled by the `IO_HIS` token; the default value is `IO_HIS = 10`.

4.8 Averaging

Set `AVERAGE = 1` in the tokens section to get averages of field variables left in files `session.ave` and `session.avg` (which are analogous to `session.chk` and `session.fld`, but `session.ave.bak` is not produced). Averages are updated every `IO_HIS` steps, and dumped every `IO_FLD` steps. Restarts are made by reading `session.avg` if it exists.

Setting `AVERAGE = 2` will accumlate averages for Reynolds stresses as well, with reserved names `ABCDEF`, corresponding to products

```
uu uv uw     A  B  D
   vv vw  =     C  E
      ww           F
```

The hierarchy is named this way to allow accumlation of products in 2D as well as 3D (for 2D you get only `ABC`). In order to actually compute the Reynolds stresses from the accumulated products you need to run the `rstress` utility, which subtracts the products of the means from the means of the products:

```
rstress session.avg > reynolds-stress.fld
```

An alternative function of `rstress` is to subtract one field file from another:

```
rstress good.fld test.fld | convert | diff
```

Setting `AVERAGE = 3` will accumlate sums of additional products for computation of terms in the energy transport equation. You will then need to use the `eneq` utility to actually compute the terms. Presently this part of the code is only written for Cartesian coordinates.

4.9 Particle tracking

The code allows for tracking of massless particles, but this only works correctly for non-concurrent execution at present. Tracking is quite an expensive operation, since Newton–Raphson iteration is used to relocate particles within each element at every timestep.

The application looks for a file called `session.par`. Each line of this file is of form

```
#     tag  time  ctime  x     y      z
      1    0.0   0.0    1.0   10.0   0.5.
```

The `time` value is the integration time, while `ctime` records the time at which integration was initialised.

Output is of the same form, and is called `session.trk`. The use of separate files, rather than by declaration in the session file, is intended so that `session.trk` files can be moved to `session.par` files for restarting. Particles that aren't in the domain at startup, or leave the domain during execution, are deleted.

Setting `SPAWN = 1`, re-initiates extra particles at the original positions every timestep. With spawning, particle tracking can quickly grow to become the most time-consuming part of execution.

Chapter 5

Compilation of specialised executables

The special compilations below can be combined.

5.1 Concurrent execution

The code supports concurrent execution for 3D simulations, with MPI used as the message-passing kernel. Compile using `make MPI=1` to produce `dns_mp`. (You will also need to compile in the appropriate message-passing routines in compiling `femlib`, for which change to the `femlib` directory, then do `make clean; make MPI=1; make install MPI=1`.) Nonlinear terms are not dealiased when running in parallel, but are dealiased in the Fourier direction when running on one process, or running the serial code. To get a serial code that does not perform dealiasing on Fourier terms (e.g. for cross-checking), compile the serial code using `make ALIAS=1` to produce `dns_alias` (in which case make sure you delete `nonlinear.o` first).

5.2 Time-varying boundary conditions

By default, the velocity boundary conditions are frozen for the time corresponding to the initial conditions. If you compile with `make TBCS=1` you will generate `dns_tbcs` in which the *zeroth Fourier mode's* (2D) velocity boundary conditions are re-evaluated every time step. Usually one only wants to vary the 2D boundary conditions. NB: make sure you delete `integrate.o` first.

5.3 Convective advection terms

By default, the code computes the nonlinear advection terms in skew-symmetric form $0.5(\boldsymbol{\nabla}\cdot\boldsymbol{u}\boldsymbol{u} + \boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{u})$, as this seems to be the most robust at high Reynolds numbers. However, it is also comparatively costly. If you want to compile a code with only the standard convective terms $\boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{u}$, compile with `make CONV=1` to produce `dns_conv`. NB: make sure you delete `nonlinear.o` first.

5.4 Stokes solver

To produce an unsteady Stokes solver (i.e. `dns` without nonlinear terms included), compile using `make STOKES=1`, to produce the executable `dns_stokes`. NB: make sure you delete `nonlinear.o` first.

Bibliography

Amon, C. H. and Patera, A. T. (1989). Numerical calculation of stable three-dimensional tertiary states in grooved-channel flow, *Phys. Fluids A* **1**(2): 2005–2009.

Blackburn, H. M. (2001). Dispersion and diffusion in coated tubes of arbitrary cross-section, *Computers and Chem. Eng.* **25**(2/3): 313–322.

Blackburn, H. M. (2002a). Mass and momemtum transport from a sphere in steady and oscillatory flows, *Phys. Fluids* **14**(11): 3997–4011.

Blackburn, H. M. (2002b). Three-dimensional instability and state selection in an oscillatory axisymmetric swirling flow, *Phys. Fluids* **14**(11): 3983–3996.

Blackburn, H. M. (2003). Computational bluff body fluid dynamics and aeroelasticity, *in* N. G. Barton and J. Periaux (eds), *Coupling of Fluids, Structures and Waves Problems in Aeronautics*, Notes in Numerical Fluid Mechanics, Springer, pp. 10–23.

Blackburn, H. M., Govardhan, R. N. and Williamson, C. H. K. (2001). A complementary numerical and physical investigation of vortex-induced vibration, *J. Fluids & Struct.* **15**(3/4): 481–488.

Blackburn, H. M. and Henderson, R. D. (1996). Lock-in behaviour in simulated vortex-induced vibration, *Exptl Thermal & Fluid Sci.* **12**(2): 184–189.

Blackburn, H. M. and Henderson, R. D. (1999). A study of two-dimensional flow past an oscillating cylinder, *J. Fluid Mech.* **385**: 255–286.

Blackburn, H. M. and Lopez, J. M. (2003a). On three-dimensional quasi-periodic Floquet instabilities of two-dimensional bluff body wakes, *Phys. Fluids* **15**(8): L57–60.

Blackburn, H. M. and Lopez, J. M. (2003b). The onset of three-dimensional standing and modulated travelling waves in a periodically driven cavity flow, *J. Fluid Mech.* **497**: 289–317.

Blackburn, H. M., Marques, F. and Lopez, J. M. (2005). Symmetry breaking of two-dimensional time-periodic wakes, *J. Fluid Mech.* **522**: 395–411.

Blackburn, H. M. and Schmidt, S. (2003). Spectral element filtering techniques for large eddy simulation with dynamic estimation, *J. Comput. Phys.* **186**(2): 610–629.

Blackburn, H. M. and Sherwin, S. J. (2004). Formulation of a Galerkin spectral element–Fourier method for three-dimensional incompressible flows in cylindrical geometries, *J. Comput. Phys.* **197**(2): 759–778.

Blackburn, H. M. and Sherwin, S. J. (2007). Instability modes and transition of pulsatile stenotic flow: Pulse-period dependence, *J. Fluid Mech.* **573**: 57–88.

Canuto, C., Hussaini, M. Y., Quarteroni, A. and Zang, T. A. (1988). *Spectral Methods in Fluid Dynamics*, Springer-Verlag, Berlin.

Deville, M. O., Fischer, P. F. and Mund, E. H. (2002). *High-Order Methods for Incompressible Fluid Flow*, Cambridge University Press.

Elston, J. R., Blackburn, H. M. and Sheridan, J. (2006). The primary and secondary instabilities of flow generated by an oscillating circular cylinder, *J. Fluid Mech.* **550**: 359–389.

Funaro, D. (1997). *Spectral Elements for Transport-Dominated Equations*, Vol. 1 of *Lecture Notes in Computational Science and Engineering*, Springer-Verlag, Berlin.

Gottlieb, D. and Orszag, S. A. (1977). *Numerical Analysis of Spectral Methods: Theory and Applications*, SIAM.

Guermond, J. L., Minev, P. and Shen, J. (2006). An overview of projection methods for incompressible flows, *Comp. Meth. Appl. Mech. & Engng* **195**: 6011–6045.

Guermond, J. L. and Shen, J. (2003). Velocity-correction projection methods for incompressible flows, *SIAM J. Numer. Anal.* **41**(1): 112–134.

Henderson, R. D. (1999). Adaptive spectral element methods for turbulence and transition, *in* T. J. Barth and H. Deconinck (eds), *High-Order Methods for Computational Physics*, Springer, chapter 3, pp. 225–324.

Henderson, R. D. and Karniadakis, G. E. (1995). Unstructured spectral element methods for simulation of turbulent flows, *J. Comput. Phys.* **122**: 191–217.

Karniadakis, G. E. (1989). Spectral element simulations of laminar and turbulent flows in complex geometries, *Appl. Num. Math.* **6**: 85–105.

Karniadakis, G. E. (1990). Spectral element–Fourier methods for incompressible turbulent flows, *Comp. Meth. Appl. Mech. & Engng* **80**: 367–380.

Karniadakis, G. E. and Henderson, R. D. (1998). Spectral element methods for incompressible flows, *in* R. W. Johnson (ed.), *Handbook of Fluid Dynamics*, CRC Press, Boca Raton, chapter 29, pp. 29–1–29–41.

Karniadakis, G. E., Israeli, M. and Orszag, S. A. (1991). High-order splitting methods for the incompressible Navier–Stokes equations, *J. Comput. Phys.* **97**(2): 414–443.

Karniadakis, G. E. and Sherwin, S. J. (2005). *Spectral/hp Element Methods for Computational Fluid Dynamics*, 2nd edn, Oxford University Press.

Korczak, K. Z. and Patera, A. T. (1986). An isoparametric spectral element method for solution of the Navier–Stokes equations in complex geometry, *J. Comput. Phys.* **62**: 361–382.

Maday, Y. and Patera, A. T. (1989). *Spectral Element Methods for the Incompressible Navier–Stokes Equations*, State-of-the-Art Surveys on Computational Mechanics, ASME, chapter 3, pp. 71–143.

Patera, A. T. (1984). A spectral element method for fluid dynamics: Laminar flow in a channel expansion, *J. Comput. Phys.* **54**: 468–488.

Rudman, M. and Blackburn, H. M. (2006). Direct numerical simulation of turbulent non-Newtonian flow using a spectral element method, *Appl. Math. Mod.* **30**(11): 1229–1248.

Sherwin, S. J. and Blackburn, H. M. (2005). Three-dimensional instabilities and transition of steady and pulsatile flows in an axisymmetric stenotic tube, *J. Fluid Mech.* **533**: 297–327.