Nek5000 Primer

This is a short primer on the basics of using Nek5000, which is based on the Nekton 2.0 spectral element code written by Paul Fischer, Lee Ho, and Einar Rønquist in 1986-1991, with technical input from A. Patera and Y. Maday. Nekton 2.0 was the first three-dimensional spectral element code and one of the first commercially available codes for distributed-memory parallel processors. The Nek5000 development was continued from the 90s through the present by P. Fischer, Jerry Kruse, Julie Mullen, Henry Tufo, James Lottes, Chaman Verma, and Stefan Kerkemeier.

1 Introduction to Nek5000

Nek5000 simulates unsteady incompressible fluid flow with thermal and passive scalar transport. It can handle general two- and three-dimensional domains described by isoparametric quad or hex elements. In addition, it can be used to compute axisymmetric flows. It is a time-stepping based code and does not currently support steady-state solvers, other than steady Stokes and steady heat conduction.

Nek5000 consists of three principal modules: the pre-processor *prex*, the solver *nek5000*, and the post-processor *postx*. Prex and postx are based upon an X-windows GUI. Nek5000 output formats can also be read by the parallel visualization package VisIt developed by Hank Childs and colleagues at LLNL/LBNL. VisIt is mandatory for large problems (e.g., more than 50,000 spectral elements). Nek5000 is written in F77 and C and uses MPI for message passing and some LAPACK routines for eigenvalue computations (depending on the particular solver employed).

In this section, we discuss the input and output files associated with the simulation engine, nek5000.

1.1 Input Data

Each simulation is defined by three files, the .rea file, the .usr file, and the SIZEu file. In addition, there is a derived .map file that is generated from the .rea file by running *genmap*. Suppose you are doing a calculation called “shear.” The key files defining the problem would be shear.rea, shear.map, and shear.usr. SIZEu controls (at compile time) the polynomial degree used in the simulation, as well as the space dimension $d = 2$ or 3.

1.1.1 *.rea* files

The file shear.rea consists of several sections:

parameters These control the runtime parameters such as viscosity, conductivity, number of steps, timestep size, order of the timestepping, frequency of output, iteration tolerances, flow rate, filter strength, etc. There are also a number of free parameters that the user can use as handles to be passed into the user defined routines in the .usr file.

logicals These determine whether one is computing a steady or unsteady solution, whether advection is turned on, etc.

geometry The geometry is specified in an arcane format specifying the xyz locations of each of the eight points for each element, or the xy locations of each of the four points for each element in 2D. (For several reasons, this format is due to be changed in the future.)

curvature This section descibes the deformation for elements that are curved.

boundary conditions Boundary conditions (BCs) are specified for each face of each element, for each *field* (velocity, temperature, passive scalar #1, etc.). The most common BC is E, which indicates that an element is connected to another element. Many of the boundary conditions support either a constant specification or a user defined specification which may be an arbitrary function. For example, a constant Dirichlet BC for velocity is specified by V, while a user defined BC is specified by v. This upper/lower-case distinction is used for all cases. There are about 70 different types of boundary conditions in all.

restart conditions Here, you can specify a file to use as an initial condition. The initial condition need not be of the same polynomial order as the current simulation. You can also specify that, for example, the velocity come from one file and the temperature from another. The initial time is taken from the restart file, but this can be overridden.

output specifications Outputs are discussed in a separate section below.

It is important to note that Nek5000 currently supports two file input formats, ascii and binary. The *.rea* file format described above is ascii. For the binary format, all sections of the .rea file having storage requirements that scale with number of elements (i.e., geometry, curvature, and boundary conditions) are moved to a second, *.re2*, file and written in binary. The remaining sections continue to reside in the *.rea* file. The distinction between the ascii and binary formats is indicated in the *.rea* file by having a negative number of elements. There are converters, reatore2 and re2torea, in the Nek5000 tools directory to change between formats. The binary file format is the default and important for i/o performance when the number of elements is large ($>$ 200000, say).

1.1.2 *.usr* files

The file shear.usr would contain set of fortran subroutines that allows the user direct access to all the variables during the course of the simulation. Here, the user may specify spatially varying properties (e.g., viscosity), volumetric heating sources, body forces, and so forth. One can specify arbitrary initial and boundary conditions through the routines *useric()* and *userbc()*. The routine *userchk()* allows the user to interrogate the solution at the end of each timestep for diagnostic purposes. Some such routines are already provided. For example, there are utilities for computing the time average of u, u^2, etc. so that one can analyze mean and rms distributions with the postprocessor. There are routine for computing the vorticity inside the solver if one is interested in tracking that, and so forth. Examples of .usr files can be found below and also with the particular case examples provided in repository.

1.1.3 SIZEu file

The SIZEu file governs the memory allocation for most of the arrays in Nek5000, with the exception of those required by the C utilities. The primary parameters of interest in SIZEu are:

ldim = 2 or 3. This must be set to 2 for two-dimensional or axisymmetric simulations (the latter only partially supported) or to 3 for three-dimensional simulations.

lx1 controls the polynomial order of the approximation, N=lx1-1.

lxd controls the polynomial order of the integration for convective terms. Generally, lxd=3*lx1/2. On some platforms, however, it is important for memory access performance that lx1 and lxd be even.

lx2 = lx1 or lx1-2. This determines the formulation for the Navier-Stokes solver (i.e., the choice between the $\mathbb{P}_N$–$\mathbb{P}_N$ or $\mathbb{P}_N$–$\mathbb{P}_{N-2}$ methods) and the approximation order for the pressure, lx2-1.

lelt determine the *maximum* number of elements *per processor*.

Memory Requirements

Per-processor memory requirements for Nek5000 scale roughly as 400 8-byte words per allocated gridpoint. The number of *allocated* gridpoints per processor is $n_{\max}$=lx1*ly1*lz1*lelt. (For 3D, lz1=ly1=lx1; for 2D, lz1=1, ly1=lx1.) If required for a particular simulation, more memory may be made available by using additional processors. For example, suppose one needed to run a simulation with 6000 elements of order $N = 9$. To leading order, the total memory requirements would be $\approx E(N+1)^3 points \times 400 (wds/pt) \times 8 bytes/wd = 6000 \times 10^3 \times 400 \times 8 = 19.2$ GB. Assuming there is 400 MB of memory per core available to the user (after accounting for OS requirements), then one could run this simulation with $P \geq 19,200$MB/(400MB/proc) = 48 processors. To do so, it would be necessary to set lelt $\geq 6000/48 = 125$.

We point out two other parameters of interest in the parallel context:

lp, the maximum number of processors that can be used.

lelg, an upper bound on the number of elements in the simulation.

There is a slight memory penalty associated with these variables, so one generally does not want to have them excessively large. It is common, however, to have lp be as large as anticipated for a given simulation so that the executable can be run without recompiling on any admissable number of processors ($P_{\text{mem}} \leq P \leq E$, where P_{mem} is the value computed above).

2 Getting Started with Meshing

Some basic programs are available to quickly generate spectral element meshes that consist of tensor-product arrays of elements. The generic name for the program is "genbox," though it has

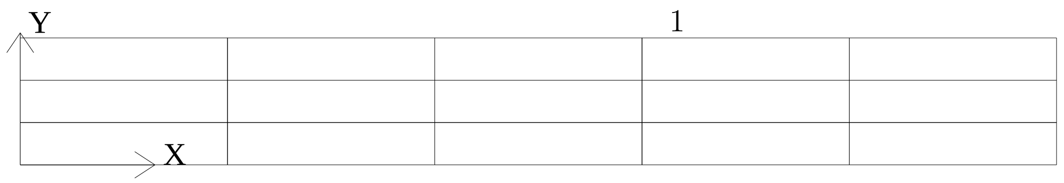

Figure 1: 5 × 3 mesh generated by *genbox*.

gone through several generations and the current version is genb6.f. It works for 2D, axisymmetric, or 3D, and can be used to couple multiple boxes together, so that the overall domain need not be a tensor-product box. In addition, any mesh built by genbox can be modified with prenek, merged with any other SE mesh by using prenek, or modified in the solver through some user-specified mesh deformation. We illustrate some of the basic concepts with the following axisymmetric example.

2.1 Uniformly Distributed Mesh

Suppose you wish to simulate flow through an axisymmetric pipe, of radius $R = 0.5$ and length $L = 4$. You estimate that you will need 3 elements in radial (y) direction, and 5 in the x direction, as depicted in Fig. 1. This would be specified by the following input file (called *pipe.box*) to genbox:

```
axisymmetric.rea
2                      spatial dimension
1                      number of fields
#=======================================================
#
#    comments:   This is the axisymmetric pipe example
#
#=======================================================
#
Box 1                         Pipe
-5 -3                         Nelx  Nely
0.0   4.0   1.0               x0  x1   ratio
0.0   0.5   1.0               y0  y1   ratio
v  ,O  ,A  ,W  ,   ,          BC's:  (cbx0, cbx1, cby0, cby1, cbz0, cbz1)
```

- The first line of this file supplies the name of an existing 2D .rea file that has the appropriate run parameters (viscosity, timestep size, etc.). These parameters can be modified later, but it is important that axisymmetric.rea be a 2D file, and not a 3D file.
- The second line indicates the number of fields for this simulation, in this case, just 1, corresponding to the velocity field (i.e., no heat transfer).
- The next set of lines just shows how one can place comments into a genbox input file.
- The line that starts with "Box" indicates that a new box is starting, and that the following lines describe a typical box input. Other possible key characters (the first character of Box, "B") are "C" and "M", more on those later.

- The first line after "Box" specifies the number of elements in the x and y directions. The fact that these values are negative indicates that you want genbox to automatically generate the element distribution along each axis, rather than providing it by hand. (More on this below.)
- The next line specifies the distibution of the 5 elements in the x direction. The mesh starts at $x = 0$ and ends at $x = 4.0$. The *ratio* indicates the relative size of each element, progressing from left to right. Here,
- The next line specifies the distibution of the 3 elements in the y direction, starting at $y = 0$ and going to $y = 0.5$. Again, *ratio*=1.0 indicates that the elements will be of uniform height.
- The last line specifies boundary conditions on each of the 4 sides of the box:
  - Lower-case v indicates that the left (x) boundary is to be a velocity boundary condition, with a user-specified distribution determined by routine *userbc* in the .usr file. (Upper-case V would indicate that the velocity is constant, with values specified in the .rea file.)
  - O indicates that the right (x) boundary is an outflow boundary – the flow leaves the domain at the left and the default exit pressure is $p = 0$.
  - A indicates that the lower (y) boundary is the axis—this condition is mandatory for the axisymmetric case, given the fact that the lower domain boundary is at $y = 0$, which corresponds to $r = 0$.
  - W indicates that the upper (y) boundary is a wall. This would be equivalent to a v or V boundary condition, with $\mathbf{u} = 0$.

2.2 Graded Mesh

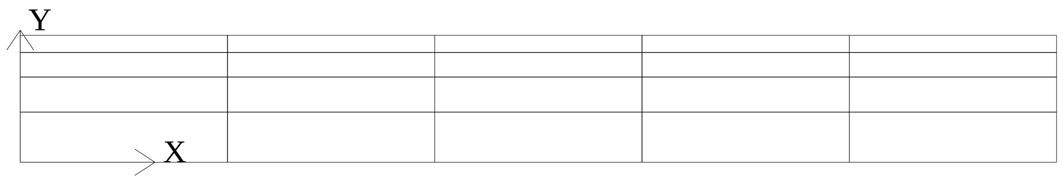

Figure 2: 5 × 2 mesh generated by *genbox*.

Suppose you wish to have the mesh be graded, that you have increased resolution near the wall. In this case you change *ratio* in the y-specification of the element distribution. For example, changing the 3 lines in the above genbox input file from

```
-5 -3                         Nelx  Nely
0.0   4.0   1.0               x0  x1   ratio
0.0   0.5   1.0               y0  y1   ratio
```

to

```
-5 -4                         Nelx  Nely
0.0   4.0   1.0               x0  x1   ratio
0.0   0.5   0.7               y0  y1   ratio
```

yields the mesh shown in Fig. 2.

2.3 User-Specified Distribution

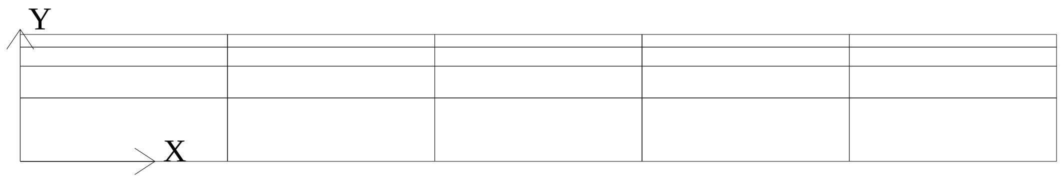

Figure 3: 5 × 4 mesh generated by *genbox*.

You can also specify your own, precise, distribution of element locations. For example, another graded mesh similar to the one of the preceding example could be built by changing the genbox input file to contain:

```
-5  4                                                 Nelx  Nely
0.0   4.0   1.0                                       x0  x1   ratio
0.000    0.250    0.375    0.450    0.500             y0  y1 ... y4
```

Here, the positive number of elements for the y direction indicates that genbox is expecting Nely+1 values of y positions on the y-element distribution line. This is the genbox default, which explains why it corresponds to Nely > 0. The corresponding mesh is shown in Fig. 3.

2.4 Mesh Modification in Nek5000

For complex shapes, it is often convenient to modify the mesh direction in the simulation code, Nek5000. This can be done throught the usrdat2 routine provided in the .usr file. The routine usrdat2 is called by nek5000 immediately after the geometry, as specifed by the .rea file, is established. Thus, one can use the existing geometry to map to a new geometry of interest.

For example, suppose you want the above pipe geometry to have a sinusoidal wall. Let $\mathbf{x} := (x, y)$ denote the old geometry, and $\mathbf{x}' := (x', y')$ denote the new geometry. For a domain with $y \in [0, 0.5]$, the following function will map the straight pipe geometry to a wavy wall with amplitude A, wavelength λ:

$$y'(x, y) = y + yA\sin(2\pi x/\lambda).$$

Note that, as $y \longrightarrow 0$, the perturbation, $yA\sin(2\pi x/\lambda)$, goes to zero. So, near the axis, the mesh recovers its original form.

In nek5000, you would specify this through usrdat2 as follows

```
c-----------------------------------------------------------------------
      subroutine usrdat2
      include 'SIZE'
      include 'TOTAL'
c
```

```
      real lambda
c
      ntot = nx1*ny1*nz1*nelt
c
      lambda = 3.
      A      = 0.1
c
      do i=1,ntot
         argx          = 2*pi*xm1(i,1,1,1)/lambda
         ym1(i,1,1,1) = ym1(i,1,1,1) + ym1(i,1,1,1)*A*sin(argx)
      enddo
c
      param(59) = 1.  ! Force nek5 to recognize element deformation.
c
      return
      end
c-----------------------------------------------------------------------
```

Note that, since nek5000 is modifying the mesh, postx will not recognize the current mesh unless you tell it to, because postx looks to the .rea file for the mesh geometry. The only way for nek5000 to communicate the new mesh to postx is via the .fld file, so you must request that the geometry be dumped to the .fld file. This is done by modifying the OUTPUT SPECIFICATIONS, which are found near the bottom of the .rea file. Specifically, change

```
  ***** OUTPUT FIELD SPECIFICATION *****
   6 SPECIFICATIONS FOLLOW
   F      COORDINATES
   T      VELOCITY
   T      PRESSURE
   T      TEMPERATURE
   F      TEMPERATURE GRADIENT
   0      PASSIVE SCALARS
```

to

```
  ***** OUTPUT FIELD SPECIFICATION *****
   6 SPECIFICATIONS FOLLOW
   T      COORDINATES                     <------  CHANGE HERE
   T      VELOCITY
   T      PRESSURE
   T      TEMPERATURE
   F      TEMPERATURE GRADIENT
   0      PASSIVE SCALARS
```

The result of above changes is shown in Fig. 4.

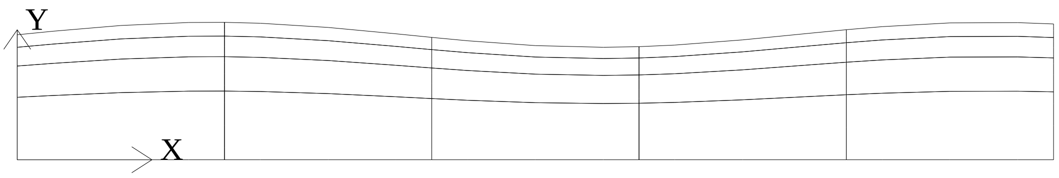

Figure 4: Wavy pipe generated by using `usrdat2` to modify the mesh of 3.

2.5 Cylindrical/Cartesian-transition Annuli (update: 7/28/04)

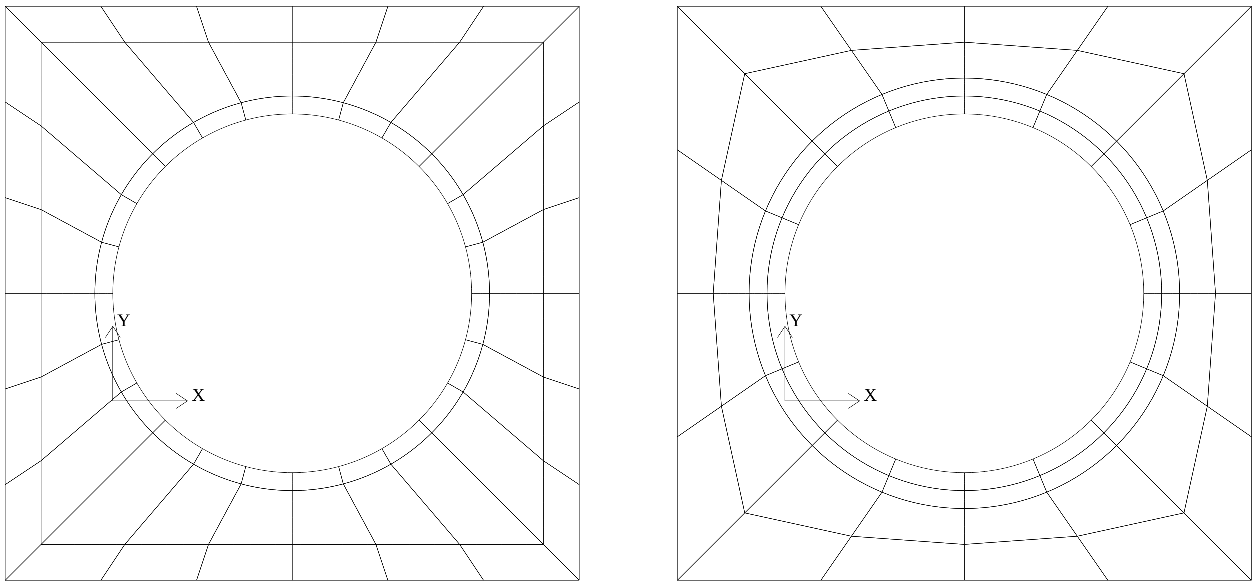

Figure 5: *Left:* Cylinder/Cartesian-transition mesh generated by *genb7* using *box7.2d* input file. *Right:* Mesh generated using *box7.2da* input file, followed by one level of quad-refine in prenek.

An updated version of genb6, known as genb7, is currently under development and designed to simply/automate the construction of cylindrical annuli, including *basic* transition-to-Cartesian elements. More sophisticated transition treatments may be generated using the GLOBAL REFINE options in prenek or through an upgrade of genb7, as demand warrants.

Example 2D and 3D input files are provided in the nek5/doc files *box7.2d* and *box7.3d.* Figure 5 shows a 2D example generated using the *box7.2d* input file, which reads:

```
x2d.rea
2                     spatial dimension
1                     number of fields
#
#    comments
#
#
#==================================================
#
Y                   cYlinder
3 -24 1             nelr,nel_theta,nelz
```

```
.5 .3                 x0,y0 - center of cylinder
ccbb                  descriptors: c-cyl, o-oct, b-box (1 character + space)
.5 .55 .7 .8          r0 r1 ... r_nelr
0  1  1               theta0/2pi theta1/2pi  ratio
v  ,W  ,E  ,E  ,      bc's (3 characters + comma)
```

An example of a mesh built with *box7.2da* is shown in Fig. 5, right. The mesh has been quad-refined once with oct-refine option of prenek. The 3D counterpart to this mesh could joined to a hemisphere/Cartesian transition built with the spherical mesh option in prenek. (See below.)

3 Flows in Periodic Domains

Nek5000 supports a wide variety of periodic boundary conditions, in all cooridinate directions. This section describes some of the principal issues involved.

3.1 Specifying Periodic Boundaries

Periodicity is specified in the .rea file by identifying pairs of element sides that are to share common degrees of freedom. A simple example would be a 2D channel flow generated using genbox, as shown below.

```
#
#    orr2d.box:   This is a 2D channel flow example
#
orr2d.rea
2                       spatial dimension
1                       number of fields
Box 1                          Channel
-3  5                          Nelx  Nely
0.0   6.28319  1.0             x0  x1   ratio ! NOTE: use usrdat2() to get 2*pi!
-1. -.80 -.35 .35 .80 1.       y0 ... y5
P  ,P  ,W  ,W  ,   ,           BC's:  (cbx0, cbx1, cby0, cby1, cbz0, cbz1)
```

Note that the genbox input uses a symmetric ordering of the faces, given in 2D as:

```
3--------[4]--------4
|                   |
|         ^         |
|        s|         |
|         |         |
[1]       +----->  [2]
```

```
|            r      |
|                   |
| (1)               |
|                   |
1--------[3]--------2
```

Thus the periodic boundary conditions above apply to faces [1] and [2] of the elements, which correspond to the x-faces for this particular mesh. On the other hand, the Nek5000 *.rea* uses a rotational vertex and faces ordering, given in 2D as:

```
 4--------[3]--------3
 |                   |
 |         ^         |
 |        s|         |
 |         |         |
[4]        +----->  [2]
 |            r      |
 |                   |
 | (1)               |
 |                   |
 1--------[1]--------2
```

The output of genbox, *box.rea*, will be in rotational form in order to adhere to the input format expected by Nek5000, prenek, and postnek.

A quick search of “ P ” in box.rea generated by genbox shows the periodic boundary conditions:

```
 P     1  4  3.000000       2.000000      0.0000000E+00 0.0000000E+00 0.0000000E+00
 P     3  2  1.000000       4.000000      0.0000000E+00 0.0000000E+00 0.0000000E+00
 P     4  4  6.000000       2.000000      0.0000000E+00 0.0000000E+00 0.0000000E+00
 P     6  2  4.000000       4.000000      0.0000000E+00 0.0000000E+00 0.0000000E+00
 P     7  4  9.000000       2.000000      0.0000000E+00 0.0000000E+00 0.0000000E+00
 P     9  2  7.000000       4.000000      0.0000000E+00 0.0000000E+00 0.0000000E+00
 P    10  4  12.00000       2.000000      0.0000000E+00 0.0000000E+00 0.0000000E+00
 P    12  2  10.00000       4.000000      0.0000000E+00 0.0000000E+00 0.0000000E+00
 P    13  4  15.00000       2.000000      0.0000000E+00 0.0000000E+00 0.0000000E+00
 P    15  2  13.00000       4.000000      0.0000000E+00 0.0000000E+00 0.0000000E+00
```

which indicates that face 4 of element 1 is periodic with element 3, face 2, and so forth.

3.2 Building Extruded Meshes with n2to3

In nek5/tools, there is a code n2to3.f that can be compiled with your local fortran compiler (preferrably not g77). By running this code, you can extend two dimensional domains to three dimensional ones with a user-specified number of levels in the z-direction. Such a mesh can then be modified using the mesh modification approach described in Section 2.4. Assuming you have a valid two-dimensional mesh, n2to3 is straightforward to run. Below is a typical session:

```
n2to3

 This is the code that establishes proper
 element connectivities, as of Oct., 2006.

 Input old (source) file name:
h2e
 Input new (output) file name:
h3e
 input number of levels: (1, 2, 3,... etc.?):
16
 input z min:
0
 input z max:
16
 input gain (0=custom,1=uniform,other=geometric spacing):
1
 This is for CEM: yes or no:
n
 Enter Z (5) boundary condition (P,v,O):
v
 Enter Z (6) boundary condition (v,O):
O
 this is cbz: v  O   <---

      320 elements written to h3e.rea
FORTRAN STOP
```

3.3 Building Spherical Meshes in PRENEK

Shperical meshes can come in many flavors. Three basic ones are the interior of a sphere, the exterior of a sphere, and a spherical shell mesh. All of these can be built with prenek. Here, we give the procedure for the latter two cases, starting with the spherical shell.

3.3.1 A Basic Spherical Shell Mesh

The following commands will build a spherical shell mesh, having inner radius 0.5 and outer radius 0.6, using 24 spectral elements.

```
preh

sph_sh
READ PREVIOUS PARAMETERS
base_ns                         (found in nek5/doc)
BUILD FROM FILE
<cr>                            (take default)

REVIEW/MODIFY

DELETE ELEMENT                  (delete single element)

CURVE SIDES
SPHERICAL MESH
0 0 0                           (coordinate of sphere center)
w                               (whole sphere)
n                               (no prolate spheroid)
s                               (spherical shell)
0.5
s                               (spherical shell)
0.6
e                               (exit spherical mesh generator)
BUILD MENU
END    ELEMENTS
ACCEPT MATL,QVOL
SET BCs
SET ENTIRE LEVEL
VELOCITY
FORTRAN FUNCTION                (set bcs to "v")
END  LEVEL
ACCEPT B.C.'s
EXIT
```

The mesh can subsequently be refined with oct- or quad-refine options in GLOBAL refine, upon reentering prenek.

3.3.2 A Spherical Shell Mesh in a Cartesian Box

The following commands will build a spherical shell mesh, having inner radius 0.5 and outer radius 0.6, using 24 spectral elements, inside a layered Cartesian box.

```
preh

sph_sh
READ PREVIOUS PARAMETERS
base_ns                        (found in nek5/doc)
BUILD FROM FILE
<cr>                           (take default)

REVIEW/MODIFY

DELETE ELEMENT                 (delete single element)

CURVE SIDES
SPHERICAL MESH
0 0 0                          (coordinate of sphere center)
w                              (whole sphere)
n                              (no prolate spheroid)
s                              (spherical shell)
0.5
s                              (spherical shell)
0.6
c                              (extend to Cartesian box)
0.9
c                              (extend to Cartesian box)
1.5
c                              (extend to Cartesian box)
2.5
c                              (extend to Cartesian box)
4.0
c                              (extend to Cartesian box)
6.5
c                              (extend to Cartesian box)
10.0
e                              (exit spherical mesh generator)
BUILD MENU
END    ELEMENTS
ACCEPT MATL,QVOL
SET BCs
SET ENTIRE LEVEL
VELOCITY
FORTRAN FUNCTION               (set bcs to "v")
END  LEVEL
ACCEPT B.C.'s
EXIT
```

The mesh can subsequently be refined with oct- or quad-refine options in GLOBAL refine, upon reentering prenek.

3.4 Combining Spherical Shell Meshes with Other Meshes

Note that you can combine a cylinder on, say, the interval [-z0,0] with a hemisphere, provided they have the same planform at z=0. To do so, you would simply build a hemisphere w/ prenek; build a cylinder + transition w/ genb7; start a new prenek session using either of these two .rea files and then IMPORT MESH to bring in the other. Prenek will automatically identify the element interface that exists at z=0 and assign the correct boundary conditions.

In a similar fashion, you can insert a sphere or hemisphere in a tensor-product array of elements by: using genbox to build a tensor product array of elements; use prenek to build a spherical region with matching Cartesian interface; using the tensor-product box to start a new session and importing the spherical region with IMPORT MESH. Prenek will query, "Displace existing elements in box?" If you reply *y* then any existing element that is in the bounding box of the incoming mesh will be deleted. This provides a convenient way of carving out a space for the new mesh within the originating tensor-product domain.

4 Data Layout

Nek5000 was designed with two principal performance criteria in mind, namely, *single-node* performance and *parallel* performance.

A key precept in obtaining good single node performance was to use, wherever possible, unit-stride memory addressing, which is realized by using contiguously declared arrays and then accessing the data in the correct order. Data locality is thus central to good serial performance. To ensure that this performance is not compromised in parallel, the parallel message-passing data model is used, in which each processor has its own local (private) address space. Parallel data, therefore, is laid out just as in the serial case, save that there are multiple copies of the arrays—one per processor, each containing different data. Unlike the shared memory model, this distributed memory model makes data locality transparent and thus simplifies the task of analyzing and optimizing parallel performance.

Some fundamentals of Nek5000's internal data layout are given below.

1. Data is laid out as $u^e_{ijk} = u(i,j,k,e)$

   `i=1,...,nx1 (nx1 = lx1)`
   `j=1,...,ny1 (ny1 = lx1)`
   `k=1,...,nz1 (nz1 = lx1` or 1, according to ndim=3 or 2)

   `e=1,...,nelv`, where `nelv` $\leq$ `lelv`, and `lelv` is the upper bound on number of elements, *per processor.*

2. Fortran data is stored in column major order (opposite of C).

3. All data arrays are thus contiguous, even when `nelv` $<$ `lelv`

4. Data accesses are thus primarily unit-stride (see chap.8 of DFM for importance of this point), and in particular, all data on a given processor can be accessed as, e.g.,

   ```
   do i=1,nx1*ny1*nz1*nelv
      u(i,1,1,1) = vx(i,1,1,1)
   enddo
   ```

   which is equivalent but superior (WHY?) to:

   ```
   do e=1,nelv
   do k=1,nz1
   do j=1,ny1
   do i=1,nx1
      u(i,j,k,e) = vx(i,j,k,e)
   enddo
   enddo
   enddo
   enddo
   ```

   which is equivalent but vastly superior (WHY?) to:

   ```
   do i=1,nx1
   do j=1,ny1
   do k=1,nz1
   do e=1,nelv
      u(i,j,k,e) = vx(i,j,k,e)
   enddo
   enddo
   enddo
   enddo
   ```

5. All data arrays are stored according to the SPMD programming model, in which address spaces that are local to each processor are private — not accessible to other processors except through interprocessor data-transfer (i.e., message passing). Thus

   ```
   do i=1,nx1*ny1*nz1*nelv
      u(i,1,1,1) = vx(i,1,1,1)
   enddo
   ```

   means different things on different processors and `nelv` may differ from one processor to the next. (By at most 1, WHY ?)

6. For the most part, low-level loops such as above are expressed in higher level routines only through subroutine calls, e.g.,:

   ```
   call copy(u,vx,n)
   ```

   where `n:=nx1*ny1*nz1*nelv`. Notable exceptions are in places where performance is critical, e.g., in the middle of certain iterative solvers.

5 Free Surface Flows

5.1 Overview of the Formulation

We consider unsteady incompressible flow in a domain with moving boundaries:

$$\frac{\partial \mathbf{u}}{\partial t} \;=\; -\nabla p \;+\; \tfrac{1}{Re}\nabla\cdot(\nabla+\nabla^T)\mathbf{u} \;+\; NL, \qquad \nabla\cdot\mathbf{u}=0, \tag{1}$$

Here, NL represents the quadratic nonlinearities from the convective term. Our spatial discretization of (1) is based on the spectral element method (SEM) [1], which is a high-order weighted residual technique similar to the finite element method. In the SEM, the solution and data are represented in terms of Nth-order tensor-product polynomials within each of E deformable hexahedral (brick) elements. Typical discretizations involve E=100–10,000 elements of order N=8–16 (corresponding to 512–4096 points per element). Vectorization and cache efficiency derive from the local lexicographical ordering within each macro-element and from the fact that the action of discrete operators, which nominally have $O(EN^6)$ nonzeros, can be evaluated in only $O(EN^4)$ work and $O(EN^3)$ storage through the use of tensor-product-sum factorization [2]. The SEM exhibits very little numerical dispersion and dissipation, which can be important, for example, in stability calculations, for long time integrations, and for high Reynolds number flows. We refer to [3] for more details.

Our free-surface hydrodynamic formulation is based upon the arbitrary Lagrangian-Eulerian (ALE) formulation described in [4]. Here, the domain $\Omega(t)$ is also an unknown. As with the velocity, the geometry $\mathbf{x}$ is represented by high-order polynomials. For viscous free-surface flows, the rapid convergence of the high-order surface approximation to the physically smooth solution minimizes surface-tension-induced stresses arising from non-physical cusps at the element interfaces, where only C^0 continuity is enforced. The geometric deformation is specified by a mesh velocity $\mathbf{w} := \dot{\mathbf{x}}$ that is essentially arbitrary, provided that $\mathbf{w}$ satisfies the kinematic condition $\mathbf{w}\cdot\hat{\mathbf{n}}|_\Gamma = \mathbf{u}\cdot\hat{\mathbf{n}}|_\Gamma$, where $\hat{\mathbf{n}}$ is the unit normal at the free surface $\Gamma(x,y,t)$. The ALE formulation provides a very accurate description of the free surface and is appropriate in situations where wave-breaking does not occur.

To highlight the key aspects of the ALE formulation, we introduce the weighted residual formulation of (1): *Find* $(\mathbf{u},p)\in X^N\times Y^N$ *such that:*

$$\frac{d}{dt}(\mathbf{v},\mathbf{u}) = (\nabla\cdot\mathbf{v},p) - \frac{2}{Re}(\nabla\mathbf{v},\mathbf{S}) + (\mathbf{v},NL) + c(\mathbf{v},\mathbf{w},\mathbf{u}), \qquad (\nabla\cdot\mathbf{u},q)=0, \tag{2}$$

for all test functions $(\mathbf{v},q)\in X^N\times Y^N$. Here (X^N,Y^N) are the compatible velocity-pressure approximation spaces introduced in [5], $(.,.)$ denotes the inner-product $(\mathbf{f},\mathbf{g}) := \int_{\Omega(t)}\mathbf{f}\cdot\mathbf{g}\,dV$, and $\mathbf{S}$ is the stress tensor $S_{ij} := \frac{1}{2}(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i})$. For simplicity, we have neglected the surface tension term. A new term in (2) is the trilinear form involving the mesh velocity

$$c(\mathbf{v},\mathbf{w},\mathbf{u}) := \int_{\Omega(t)}\sum_{i=1}^{3}\sum_{j=1}^{3} v_i\frac{\partial w_j u_i}{\partial x_j}\,dV, \tag{3}$$

which derives from the Reynolds transport theorem when the time derivative is moved outside the bilinear form $(\mathbf{v},\mathbf{u}_t)$. The advantage of (2) is that it greatly simplifies the time differencing and avoids grid-to-grid interpolation as the domain evolves in time. With the time derivative outside of the integral, each bilinear or trilinear form involves functions at a specific time, t^{n-q}, integrated over $\Omega(t^{n-q})$. For example, with a second-order backward-difference/extrapolation scheme, the discrete form of (2) is

$$\frac{1}{2\Delta t}\left[3(\mathbf{v}^n,\mathbf{u}^n)^n - 4(\mathbf{v}^{n-1},\mathbf{u}^{n-1})^{n-1} + (\mathbf{v}^{n-2},\mathbf{u}^{n-2})^{n-2}\right] = L^n(\mathbf{u}) + 2\widetilde{NL}^{n-1} - \widetilde{NL}^{n-2}. \qquad (4)$$

Here, $L^n(\mathbf{u})$ accounts for all *linear* terms in (2), including the pressure and divergence-free constraint, which are evaluated implicitly (i.e., at time level t^n, on $\Omega(t^n)$), and $\widetilde{NL}^{n-q}$ accounts for all *nonlinear* terms, including the mesh motion term (3), at time-level t^{n-q}. The superscript on the inner-products $(.,.)^{n-q}$ indicates integration over $\Omega(t^{n-q})$. The overall time advancement is as follows. The mesh position $\mathbf{x}^n \in \Omega(t^n)$ is computed explicitly using $\mathbf{w}^{n-1}$ and $\mathbf{w}^{n-2}$; the new mass, stiffness, and gradient operators involving integrals and derivatives on $\Omega(t^n)$ are computed; the extrapolated right-hand-side terms are evaluated; and the implicit linear system is solved for $\mathbf{u}^n$. Note that it is only the *operators* that are updated, not the *matrices.* Matrices are never formed in Nek5000 and because of this, the overhead for the moving domain formulation is very low.

5.2 Developing Free Surface Applications

Nek5000 supports several free-surface options that were developed by Lee Ho as part of his Ph.D. thesis and while he was a principal at Nektonics. Because it is not heavily used by the current development group, users of the free-surface features in Nek5000 will have to develop their own experience with the capabilities and limitations of the current formulation. Additions to this write-up are welcomed and encouraged.

There are three steps to setting up a free-surface calculation in Nek5000:

- In the .rea file: set IFSTRS and IFMVBD to T (true); set MS (or other) conditions on the free-surface boundaries; and set IFXYO=T to ensure output of the geometry.
- In the SIZEu file, set `lx1m=lx1, ly1m=ly1, lz1m=lz1` and recompile all object files to allocate memory to the free-surface variables.
- In the .usr file, set surface tension (if not done through bcs).

We introduce a couple of examples here.

5.2.1 Flow Down an Plate

We have extracted the following test case from Lee Ho's thesis [4], which also provides reference to experimental data.

A viscous coating on a vertical plate is subjected to the force of gravity. After a short distance (here, time), waves develop with a given shape and wavelength. The relevant parameters are:

- Weber number: $We := x$
- Reynolds number: $Re := x$

6 History Points

```
      0 PACKETS OF DATA FOLLOW
 ***** HISTORY AND INTEGRAL DATA *****
          56   POINTS.  Hcode, I,J,H,IEL
UVWP     H     31   31    1    6
UVWP     H     31   31   31    6
UVWP     H     31   31   31   54
UVWP     H     31   31   31  102
UVWP     H     31   31   31  150
 "       "      "    "    "    "
 "       "      "    "    "    "
```

Above some are some lines that define history points in the .rea file. Some important comments:

1. The "56 POINTS" line needs to be followed by 56 lines of the type shown. Where the "56" is on that line doesn't matter (as long as it is first in the line, the rest is ignoored...)
2. However, in each of the following lines, which have the UVWP etc., location is CRUCIAL. It must be layed out *exactly* as indicated above. (Why? Because those lines contain character strings, they use formatted reads, which are very particular about where the data is. Lines that do *not* have character entries can use free formatted reads, where blanks serve as separators; thus, no format restrictions.)
3. So, if you want to pick points close to the center of element 1 and are running with lx1=10, say, you might choose:

   ```
   UVWP     H      5    5    5    1
   ```

   which I created by copy-paste of a line above and then overwrote while preserving the right-adjusted integer locations. (In fact, the indicated point would really be at the middle of the element only if lx1=9 in this case. Why?)
4. The UVWP tells the code to write the 3 velocity components and pressure to the .sch file at each timestep (or, more precisely, whenever mod(istep,iohis)=0, where iohis=param(52)).

   Note that if you have more than one history point then they are written sequentially at each timestep. Thus 10 steps in the first example with param(52)=2 would write (10/2)*56 = 280 lines to the .sch file, with 4 entries per line.
5. The "H" indicates that the entry corresponds to a requested history point.
6. If the ijk values (5 5 5 in the preceding example line) exceed lx1,ly1,lz1 of your SIZEu file, then they are truncated to that value. For example, if lx1=10 for the data at the top (31 31 31) then the code will use ijk of (10 10 10), plus the given element number, in identifying the history point.

   It is often useful to set ijk to large values (i.e., > lx1) because the endpoints of the spectral element mesh are invariant when lx1 is changed. Thus, because "1" is also invariant, the entry

```
 UVWP     H     31    1    1    6
```

would track the same point in physical space for all values of lx1 up to lx1=31.

7. A difficulty with the current nek history point specification is finding the requisite ijke (e=element number) values that correlate to the point of interest. There is a way to do this in postx that is relatively painless, but this is not useful for very large problems. (The approach is:

   ```
   SET PLOT FORMAT
   SCALAR
   VALUES
   PLOT
   ```

   Follow the instructions and for each point requested, postx will write to the screen lines that are similar to the above, ready to be pasted into the .rea file.)

8. When you run nek, it will write the coordinate information to the logfile on the first timestep so that you can verify the point locations.

7 README FILES

Below are the contents of the README file in the nek5/doc directory. There are several others there as well, so some answers to questions may be found in one or more of those.

```
January 20,2002 pff

o  drive.f now supports constant volume flow rate in x, y, or z
   direction, depending on whether param54 is 1,2,or 3.  x=default
   for any other value of p54.

o  Global fdm method incorporated as solver and preconditioner for
   E solve when p116=nelx, p117=nely, p118=nelz.   Note that setting
   one of these to its negative value determins the "primary" direction
   that is wholly contained within each processor.  Thus, if, for example,
   nelx = -6, nely = 8 and nelz = 5, then the 8x5 array of elements would
   be partitioned among P processors, and each processor would receive
   (8x5)/P stacks of depth 6.   It is thus relatively important that
   the product of the number of elements in the remaining secondary
   and tertiary directions (those not flagged by a minus sign) should
   be a multiple of P.

o  Dealiasing is currently enabled, whenever p99=2.   There is some memory
   savings to be had if dealiasing is not being used by editing DEALIAS
```

and commenting out the appropriate parameter statements.

o The comm_mpi routine has been cleaned up.   All vector reductions are performed using mpi_all_reduce so, in theory, there should not be a constraint that P=2**cd, provided one isn't using the XXt solver (e.g., if one is using the gfdm solver).

o The XXt solver is almost in place for the steady conduction case.

9/10/01, pff

The routines navier5.f and connect1.f have been modified to allow for multiple passive scalars.

Nekton consists of three principal modules: the pre-processor, the solver, and the post-processor. The pre- and post-processors, (resp. prenek and postnek) are based upon an X-windows GUI.   Recently, postnek has been extended to output Visual Toolkit (vtk) files, so that output my be viewed in the cave. This is still under development.

The solver, nek5000, is written in F77 and C and supports either MPI or Intel's NX message passing libraries.   It is a time-stepping based code and does not currently support steady-state solvers, other than steady Stokes and steady heat conduction.

To get started, you create a directory, say, nek5, and unpack the tarfile:

```
mkdir  nek5
mv tarfile.gz nek5
cd nek5
gunzip tarfile
tar -xvf tarfile
rm tarfile
```

The tarfile will put the source code in the directory src, and will create additional subdirectories: nek5/bin, nek5/2d8, nek5/3d6, and nek5/prep.

nek5/bin contains the shell scripts which I use to start a job.
I usually keep these files in my /bin directory at the top level.
The scripts will be of use so you can see how I manage the files
created by nek5.

nek5/3d6 (2d8) contains an example .rea file, with associated .fld00 file
for restarting. Once you "gunzip *" in all of the directories
you should be able to build the executable by typing "makenek" in
the 3d6 directory.

For each job that you run, you need a corresponding "subuser.f" file in the
nek5/src directory. "subuser.f" provides the user supplied function
definitions (e.g., for boundary conditions, initial conditions, etc). I
usually keep several subuser.f files in the nek5 directory for different
cases under the name of *.user, e.g., channel.user for channel flow, or
cyl.user for flow past a cylinder, etc.

To build the source, simply

```
cd 3d6
makenek   (or,  "makebk" to make in the background...)
```

To run it, type

```
nekb co1a
```

This will run the job entitled "co1a" in background. The scripts I have
set up in nek5/bin are:

```
   nek:    runs interactively,

   nekl:   runs interactively, but redirects std. out to job.log ( & logfile)

   nekb:   like nekl, but in the background

   nekd:   like nek, but with dbx
```

Each looks for the session name (the prefix of the "*.rea" file) as
an argument.

To build the pre- and post-processors, you will cd to nek5/prep.
Type "make" to make postx, and "make -f mpre" to build prex.

If you then put postx and prex into your top-level bin directory
you'll be able to invoke the pre and post-processors with the
commands "prex" or "postx", etc. (I assume you're already somewhat
familiar w/ using pre and post?).

```
--------------------------------------------------------------


Here's a brief explanation of the nek5000 input format.

First, an overview of the structure of the file:

Section I:   Parameters, logical switches, etc.

  This section tells nek5000 whether the input file reflects
  a 2D/3D job, what combination of heat transfer/ Stokes /
  Navier-Stokes/ steady-unsteady / etc.  shall be run.

  What are the relevant physical parameters.
  What solution algorithm within Nekton to use, what timestep size
  or Courant number to use, or whether to run variable DT, etc.


Section II:   Mesh Geometry Info

  This section gives the number of elements, followed by the
  4 (8) vertex pairs (triplets) which specify the corner of
  each two- (three-) dimensional element.

  A subsection which follows specifies which element surfaces
  a curved.  The exact parameter definitions vary according to
  the type of curved surface.  I usually make my own definitions
  when I need to generate a new curved surface, e.g., spheres
  or ellipsoids.  We use the Gordon-Hall mapping to generate
  the point distribution within elements, and map according to
  arc-length for points along the edges of two-dimensional
  elements.  For 3D spheres, I take the element surface point
  distribution to be the intersection of the great circles
  generated by the corresponding edge vertices, and again use
  Gordon-Hall to fill the element interior volume.

  The next subsection contains all the boundary condition information
  for each element.  "E" means that the element is connected to
  the element specifed by parameter 1, on the side specified by
  parameter 2.  "T" means a constant temperature along the edge
  of the element, "t" means a temperature distribution according
  to a fortran function which the user writes and links in with
  source code contained in "subuser.f".   Similarly, "V" means
  a constant velocity along the element edge (three components
  specifed in parameters 1, 2, and 3, resp.), and "v" implies a
  user specifed fortran function.  "O" is for outflow, "SYM" is
  for symmetry bc's., etc.

Section III:   Output Info.
```

  This section specifies what data should be output, including
  field dumps (i.e., X,Y,Z + U,V,W + P + T, or any combination
  thereof) and time history trace info... e.g., u(t), v(t), etc.
  at a particular point, for each time step. (Corresponds to a
  hot wire probe output in some sense.)

  Also, if this run is to restart from a previous field dump (e.g.,
  one computed with a lower order, but the same spectral element
  configuration), that is specified in this section.

This is very brief, but it should give you a road map to the
general layout of the input files.

Using PRENEK

This is a brief description of how to modify an existing mesh.
Typically it's easiest to modify run parameters just by editing
the file. If you wish to modify the geometry, it's (generally)
best to do this through prenek. To do this, type "pre"
(to begin executing prenek).

Then, click on "READ PREVIOUS PARAMETERS"

Enter "box" with the keyboard.

Then "ACCEPT PARAMETERS"

Then, "BUILD FROM FILE"

Just hit <cr> to accept the default, box.rea.

Then, ACCEPT... ACCEPT... ACCEPT... EXIT

Using PRENEK to set periodic BCs

Currently, genb6 will properly set periodic BCs only for the single
box case, but not for geometries that feature multiple boxes defined
in a single file. (If someone wants to volunteer to fix this, assistance
is always welcomed.) In this case, it's generally easiest to set the
periodic bcs using the PERIODIC AUTO feature in prenek.

To do this:

```
you are correct -- multibox + genbox will not generate correct
                   periodic bcs (in general).

The best bet is to leave the sides that you wish to be periodic
blank, then to run the mesh through prenek as follows.

Start prex.

Click on "READ PREVIOUS PARAMETERS"

Type "box"  using the keyboard.

Click "ACCEPT PARAMETERS"

Click, "BUILD FROM FILE"

Hit <cr> to accept the default, box.rea.

Click,  ACCEPT...  ACCEPT... until you get to the
SET BCs menu and click on that.

Click on SET ENTIRE LEVEL (near the bottom).
Click on PERIODIC AUTO

answer "n" to the question at the bottom of the screen.

Then click ACCEPT ACCEPT etc. as you move to exit.

Run genmap to generate the requisite .map file.


-------------------------------------------------------------------------

The .rea file can be edited to change

    viscosity
    number of steps
    Courant number (typ.=2)
    Torder (order of time stepping - typ. 1 in the beginning,
            for more stability, then Torder=2 when you are after
            sensitive results...)
    IOSTEP  (frequency of .fld dumps)
    etc.


==============================================================================
STARTING THE JOB:
```

```
If you've compiled your code (with LELT=LELV=sufficiently large), you
should now be set to run nek5000.  For serial simulations, in nek5/bin
there are several scripts for running.  To use them to run, say blah.rea,
type:

../bin/nek   blah    -- runs interactively,

../bin/nekl  blah    -- runs interactively, but redirects stdout to blah.log ( & logfile

../bin/nekb  blah    -- like nekl, but in the background

../bin/nekd  blah    -- like nek, but starts gdb for debugging


A successful run will produce blah.fld01 ... blah.fldnnn, where nnn
is the number of .fld files determined by your particular combination
of NSTEPS, FINTIME, IOSTEP, IOTIME, or extra calls to outpost() that
might be in blah.usr.


==============================================================================
ENDING THE JOB:

   The job will quit after either FINTIME is reached (I never use this)
   or NSTEPS have completed.

   Should you wish to terminate sooner, or get an output *right now*, I've
   implemented a little check for a file called "ioinfo" into the code.

      To get an output at the end of the current step, but keep on running,
      type:  "echo 1 > ioinfo".

      To get an output at the end of the current step, and then stop the job,
      type:  "echo -1 > ioinfo".

      To stop the job without dumping the .fld file,
      type:  "echo -2 > ioinfo".
==============================================================================

CHANGING BOUNDARY CONDITIONS, INITIAL CONDITIONS, ETC.

I've set the genbox2.dat and genbox2.small files to specify fortran boundary
conditions at inflow (denoted by lower-case characters)

These are computed in the user accessible source code
in "../src/subuser.f"  (SUBROUTINE USERBC and SUBROUTINE USERIC).

In the subuser.f file which I gave you,  I specified
a blasius profile for the initial and boundary conditions.
```

This is the same code I used for my hemisphere runs

NOTE*

You probably will want to change the boundary layer thickness (delta) in subuser.f. Just edit subuser.f, search for where delta is specified, change it, and go back to 3d6 (3d8? 3d10?) and type "makenek". We can very easily make this a runtime parameter, if you find that you're making many trials with different boundary layer thicknesses.

==

RESTARTING

This is really easy...

Just take the .rea file you're about to start with, go to the bottom of the file, go up ~33 lines, and change the line:

```
"            0 PRESOLVE/RESTART OPTIONS  *****"
```

to:

```
"            1 PRESOLVE/RESTART OPTIONS  *****"
"my_old_run"
```

or:

```
"            1 PRESOLVE/RESTART OPTIONS  *****"
"my_old_run.fld"
```

or:

```
"            1 PRESOLVE/RESTART OPTIONS  *****"
"my_old_run.fld01  TIME=0.0 "
```

etc.... (note, drop quotations)

Note that the new run must have the same topology as the old run, that is, the same number of elements in the same location. However, you can change polynomial degree by running in a different directory (e.g., 3d8) with a code which has been compiled with a different SIZEu file.

Files needed to start a particular run called, say, "hemi1":

```
    hemi1.rea
    hemi1.sep
    hemi1.map
```

If hemi1 is to use an old field file (say, "hemi_old.fld23") as an initial condition, you will also need to have:

```
    hemi_old.fld23
```

If hemi_old.fld23 is in binary format, you will need the associated
header file (which is ascii):

```
    hemi_old.fhd23
```

Note that hemi_old must have the same number of elements as hemi1.
However, it can be of different polynomial degree, provided it is
of degree less than or equal to NX1+2, where NX1 is the number
of grid points in each direction within each element.
(Note: NX1 := polynomial degree + 1.)

In addition to the above hemi1-specific files, you will also need
a job specific "subuser.f" file to be compiled and linked in with
the source code. Frequently this file is the same over a broad
range of parametric studies. It incorporates the user defined
forcing functions, boundary conditions, and initial conditions (if
not restarting from a previous .fld file). Hence, it's likely
that you will not need to edit/recompile this routine very often.

Note that if you are simultaneously undertaking two *different*
Nekton studies in two different directories which both point to
the same source, you will need to swap subuser.f files each time
you recompile the source in either directory. I usually do this
by keeping the subuser.f files in the src directory under the names,
e.g., hemi.user, cylinder.user, channel.user, etc. Then when I'm
compiling the source code for the hemisphere problem I would go to
the /src directory and copy hemi.user to subuser.f prior to compiling
in the working directory.

To build nek5000,

```
cd 3d6
makenek
```

To run nek5000 in foreground

```
../bin/nek hemi1
```

To run nek5000 in background

```
../bin/nekb hemi1
```

To terminate a job (other than "kill")

```
echo  1 > ioinfo        --- dumps fld file after current step and continues

echo -1 > ioinfo        --- dumps fld file at end of step and quits

echo -2 > ioinfo        --- quits at end of step. No fld file dumped.

echo -# > ioinfo  (#>2) --- quits at end of step #, after dumping fld file.
```

To run with multiple passive scalars (e.g., temperature and one or more other convected fields), you should do the following.

0) Make sure that you have a version of nek5000 and postx that works with multiple passive scalars. (If you receive these after 9/9/01, you probably do.)

1) Assuming you already have a .rea file set for Navier-Stokes plus heat transfer, you can use prex to generate a new .rea/.map/.sep file set that allows for an additional passive scalar. Simply start prenek in the director of interest. Read parameters from your existing .rea file, then select

```
     ALTER PARAMETERS

     PASSIVE SCALAR    (1)

     WITH CONVECTION   (Y)
```

   Hit <cr> through all the parameters (edit these in the .rea file later, with the editor of your choice). Then, read the geometry from your existing file. When you get to the boundary condition menu, you will be prompted for BC's for the new passive scalar. SET ENTIRE LEVEL, Insulated, is a common choice.

2) Exit prenek. Note that the default conductivity and rhocp for the new passive scalar is (1.0,1.0). These can be changed by editing the .rea file (if you're not making them functions of time or space). Simply locate the following lines (found right after the parameters)

```
      4  Lines of passive scalar data follows2 CONDUCT; 2RHOCP
   0.00100       1.00000       1.00000       1.00000       1.00000
```

```
1.00000      1.00000      1.00000      1.00000
2.00000      1.00000      1.00000      1.00000      1.00000
1.00000      1.00000      1.00000      1.00000
```

```
The first 2 lines are the conductivities for the 9 passive scalars.
You only need to set the first of these.  In the example above,
we have conduct(PS1) =.001.

The second 2 lines are the rhocp values for the 9 passive scalars.
You only need to set the first of these.  In the example above,
we have rhocp(PS1) =2.0.
```

An example of specifying surface normals in the .usr file

```
c-----------------------------------------------------------------------
      subroutine userbc (ix,iy,iz,iside,eg)
      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'

      integer e,eg,f
      real snx,sny,snz   ! surface normals

      f = eface1(iside)
      e = gllel (eg)

      if (f.eq.1.or.f.eq.2) then      ! "r face"
         snx = unx(iy,iz,iside,e)                 ! Note:  iy,iz
         sny = uny(iy,iz,iside,e)
         snz = unz(iy,iz,iside,e)
      elseif (f.eq.3.or.f.eq.4)  then ! "s face"
         snx = unx(ix,iz,iside,e)                 !        ix,iz
         sny = uny(ix,iz,iside,e)
         snz = unz(ix,iz,iside,e)
      elseif (f.eq.5.or.f.eq.6)  then ! "t face"
         snx = unx(ix,iy,iside,e)                 !        ix,iy
         sny = uny(ix,iy,iside,e)
         snz = unz(ix,iy,iside,e)
      endif

      ux=0.0
      uy=0.0
      uz=0.0
      temp=0.0
      return
      end
c-----------------------------------------------------------------------
```

Some comments on Courant number

```
A bit about COURANT in the .rea file.

It serves different roles, depending on the choice
of timestepping.

For std. timestepping (BDF/EXT), with DT < 0, it does nothing.

For std. timestepping (BDF/EXT), with DT > 0, it determines
the upper bound for dt.  So, if you say DT=0.5 and COURANT=0.1,
then it will choose dt to be the min. of (DT,dt_cfl), where
dt_cfl is the value of dt such that CFL=0.1.  (There is one
slight variance to this...but essentially this is what happens.)

For std. timestepping (BDF/EXT), with DT = 0, it determines
dt to be dt_cfl.

If IFCHAR=.true. (i.e., OIFS), then COURANT determines how
many substeps you will take to achieve the desired DT value
(which is specified by param(12)).  So, if you say COURANT=12,
it would take 12 RK4 substeps ( --> 48*k convection evaluations,
where k=torder) to advance each field.  Thus, you don't want
COURANT to be too much larger than the CFL that is reported
on the "Step" line in the logfile.

TO SUMMARIZE, for OIFS:

Set param 12 = -.001   (say, if that's desired)

Set COURANT = 3        (say, if dt=.001 gives a cfl of ~3)



[ One more thing:  For std. timestepping, if you set COURANT > 0.5,
                   then nekton will automatically reset to a lower
                   value -- 0.25 or 0.4.... something like that...]
```

Some comments on history points

History points allow one to trace the time history (e.g., $\mathbf{u}(\mathbf{x}_i, t)$) at selected grid points $\mathbf{x}_i$. Below are some lines that define history points in the .rea file.

```
      0 PACKETS OF DATA FOLLOW
  ***** HISTORY AND INTEGRAL DATA *****
           56   POINTS.  Hcode, I,J,H,IEL
 UVWP      H      31   31    1    6
 UVWP      H      31   31   31    6
 UVWP      H      31   31   31   54
 UVWP      H      31   31   31  102
 UVWP      H      31   31   31  150
  "        "       "    "    "    "
  "        "       "    "    "    "
```

```
Some important comments:

1.  The "56 POINTS" line needs to be followed by 56 lines
    of the type shown.


2.  Where the "56" is on that line doesn't matter (as long as it
    is first in the line, the rest is ignoored...)

3.  However, in each of the following lines, which have the
    UVWP etc., location is CRUCIAL.   It must be layed out
    _exactly_ as indicated above.   (Why?  Because those lines
    contain character strings, they use formatted reads, which
    are very particular about where the data is.  Lines that
    do _not_ have character entries can use free formatted reads,
    where blanks serve as separators; thus, no format restrictions.)

    So, if you want to pick points close to the center of element 1
    and are running with lx1=10, say, you might choose:

 UVWP      H       5    5    5    1

    which I created by copy past of a line above and then overwrote
    while preserving the right-adjusted integer locations.  (In fact,
    the indicated point would really be at the middle of the element
    only if lx1=9 in this case.  Why?)

4.  The UVWP tells the code to write the 3 velocity components and
    pressure to the .sch file at each timestep (or, more precisely,
    whenever mod(istep,iohis)=0, where iohis=param(52)).
```

Note that if you have more than one history point then they are
write sequentially at each timestep.

5. The "H" indicates that the entry corresponds to a requested history point.

6. If the ijk values (5 5 5 in the preceding example line) exceed lx1,ly1,lz1 of your SIZEu file, then they are truncated to that value. For example, if lx1=10 for the data at the top (31 31 31) then the code will use ijk of (10 10 10), plus the given element number, in identifying the history point.

   It is often useful to set ijk to large values (i.e., > lx1) because the endpoints of the spectral element mesh are invariant when lx1 is changed. Thus, the entries

```
 UVWP     H      1   31    1    6
 UVWP     H     31    1    1    6
```

   would track the same points in physical space for all values of lx1 up to lx1=31.

7. The difficulty with the current nek history point specification is finding the requisite ijke (e=element number) values that correlate to the point of interest.

   There is a way to do this in postx that is relatively painless, but this is not useful for very large problems. (The approach is:

```
SET PLOT FORMAT
SCALAR
VALUES
PLOT
```

   follow the instructions and for each point requested, postx will write to the screen lines that are similar to the above, ready to be pasted into the .rea file.)

8. When you run the code, nek will write the coordinate information to the logfile on the first timestep so that you can verify the point locations.

8 Recent Updates

01/2002 drive.f now supports constant volume flow rate in x, y, or z direction, depending on whether param54 is 1,2,or 3. x=default for any other value of p54.

01/2002 Global fdm method incorporated as solver and preconditioner for E solve when p116=nelx, p117=nely, p118=nelz. Note that setting one of these to its negative value determins the "primary" direction that is wholly contained within each processor. Thus, if, for example, nelx = -6, nely = 8 and nelz = 5, then the 8x5 array of elements would be partitioned among P processors, and each processor would receive (8x5)/P stacks of depth 6. It is thus relatively important that the product of the number of elements in the remaining secondary and tertiary directions (those not flagged by a minus sign) should be a multiple of P.

01/2002 Dealiasing is currently enabled, whenever p99=2. There is some memory savings to be had if dealiasing is not being used by editing DEALIAS and commenting out the appropriate parameter statements.

01/2002 The comm_mpi routine has been cleaned up. All vector reductions are performed using mpi_all_reduce so, in theory, there should not be a constraint that P=2**cd, provided one isn't using the XXt solver (e.g., if one is using the gfdm solver).

01/2002 The XXt solver is almost in place for the steady conduction case.

01/2002 The routines navier5.f and connect1.f have been modified to allow for multiple passive scalars. (9/10/01). 9/10/01, pff

9 Ordering Data on Faces

```
Points in the volume are stored in lexicographical order:

       real u(lx1,ly1,lz1)

implies that u(i+1,j,k) follows u(i,j,k) in memory, so the
data layout (in memory) is:


   u(1,1,1) u(2,1,1)...u(lx1,1,1) u(1,2,1) u(2,2,1)... u(lx1,ly1,lz1)

For u(i,j,k) the i index is associated with r, j with s, and k with t.

There are two numberings of the faces.  The one that you're
most likely to interact with, and which is used for assigning
bcs, unit normals, surface Jacobians, etc. is what we refer to
as the "preprocessor notation".   In 2D, preprocessor notation
uses the counter-clockwise ordering:


                        ^ s
                        |   Face 3
                  +-----------+
                  |     |     |
                  |     |     |
          Face 4  |     +-----|--> r
                  |           |
                  |           |  Face 2
                  +-----------+
                     Face 1
```

In 3D, it’s the same, save that you also have Face 5 assocated with
t=-1 and Face 6 associated with t=+1.

When you traverse a face, the indices describing that 2D manifold
advance in positive order and in that case the associated face
objects (e.g., unit normals and area) will have unit stride.

There are two ways to stride across a face f=1,...,6 (and there
are many routines implementing these).

Here is one:

```
       integer e,f

       call facind (i0,i1,j0,j1,k0,k1,f)

       ia   = 0
       flux = 0
       do k=k0,k1
       do j=j0,j1
       do i=i0,i1
          ia = ia + 1
          flux = flux + ( vx(i,j,k,e)*unx(ia,1,f,e)
      $                   vy(i,j,k,e)*uny(ia,1,f,e)
      $                   vz(i,j,k,e)*unz(ia,1,f,e) )*area(ia,1,f,e)
       enddo
       enddo
       enddo
```

Here is another:

```
       call facind2 (js1,jf1,jskip1,js2,jf2,jskip2,f)
       ia = 0
       flux = 0
       do j2=js2,jf2,jskip2
       do j1=js1,jf1,jskip1
          ia = ia+1
          flux = flux + ( vx(j1,j2,1,e)*unx(ia,1,f,e)
      $               +   vy(j1,j2,1,e)*uny(ia,1,f,e)
      $               +   vz(j1,j2,1,e)*unz(ia,1,f,e) )*area(ia,1,f,e)
       enddo
       enddo
```

I like the second approach because it is faster.

Note that unx,uny,unz are _face_ based structures, while vx
vy vz are volumetric.

10 About COURANT in the .rea file

The COURANT parameter serves different roles, depending on the choice
of timestepping.

For std. timestepping (BDF/EXT), with DT < 0, it does nothing.

For std. timestepping (BDF/EXT), with DT > 0, it determines
the upper bound for dt. So, if you say DT=0.5 and COURANT=0.1,
then it will choose dt to be the min. of (DT,dt_cfl), where
dt_cfl is the value of dt such that CFL=0.1. (There is one
slight variance to this...but essentially this is what happens.)

For std. timestepping (BDF/EXT), with DT = 0, it determines
dt to be dt_cfl.

If IFCHAR=.true. (i.e., OIFS), then COURANT determines how
many substeps you will take to achieve the desired DT value
(which is specified by param(12)). So, if you say COURANT=12,
it would take 12 RK4 substeps (--> 48*k convection evaluations,
where k=torder) to advance each field. Thus, you don't want
COURANT to be too much larger than the CFL that is reported
on the "Step" line in the logfile.

TO SUMMARIZE, for OIFS:

```
Set param 12 = -.001   (say, if that's desired)

Set COURANT = 3        (say, if dt=.001 gives a cfl of ~3)
```

```
[ One more thing:  For std. timestepping, if you set COURANT > 0.5,
                   then nekton will automatically reset to a lower
                   value -- 0.25 or 0.4.... something like that...]
```

References

[1] A.T. Patera. A spectral element method for fluid dynamics : laminar flow in a channel expansion. *J. Comput. Phys.*, 54:468–488, 1984.

[2] S.A. Orszag. Spectral methods for problems in complex geometry. *J. Comput. Phys.*, 37:70–92, 1980.

[3] M.O. Deville, P.F. Fischer, and E.H. Mund. *High-order methods for incompressible fluid flow.* Cambridge University Press, Cambridge, 2002.

[4] L.W. Ho. *A Legendre spectral element method for simulation of incompressible unsteady viscous free-surface flows.* PhD thesis, Massachusetts Institute of Technology, 1989. Cambridge, MA.

[5] Y. Maday and A.T. Patera. Spectral element methods for the Navier-Stokes equations. In A.K. Noor and J.T. Oden, editors, *State-of-the-Art Surveys in Computational Mechanics*, pages 71–143. ASME, New York, 1989.