ANL/MCS-TM-351

Nek5000 User Documentation

Mathematics and Computer Science Division

About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The LaboratoryâĂŹs main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

Availability of This Report:

This report is available, at no cost, at http://www.osti.gov/bridge. It is also available on paper to the U.S. Department of Energy and its contractors, for a processing fee, from:
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
phone (865) 576-8401
fax (865) 576-5728
reports@adonis.osti.gov

ANL/MCS-TM-351

Nek5000 User Documentation

by:
Paul Fischer et al.

Contents

Executive Summary

Abstract here.

Chapter 1

Introduction

Nek5000 is designed to simulate laminar, transitional, and turbulent incompressible or low Mach-number flows with heat transfer and species transport. It is also suitable for incompressible magnetohydrodynamics (MHD). Nek5000 is written in f77 and C. It uses MPI for message passing (but can be compiled without MPI for serial applications) and some LAPACK routines for eigenvalue computations (depending on the particular solver employed). In addition, it can be optionally coupled with MOAB, which provides an interface to meshes generated with CUBIT. Nek5000 output formats can be read by either `postx` or the parallel visualization package VisIt developed by Hank Childs and colleagues at LLNL/LBNL. VisIt is mandatory for large problems (e.g., more than 100,000 spectral elements).

Computational approach

The spatial discretization 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. The code Nek5000 is based on the following design principles

- accessible both to beginners and experts alike
- accessible interface via Fortran to include user-defined modules
- the code intrinsics can be accessed and modified via the user files for more experienced developers
- portability
- minimal use of external libraries to assure fast compile times

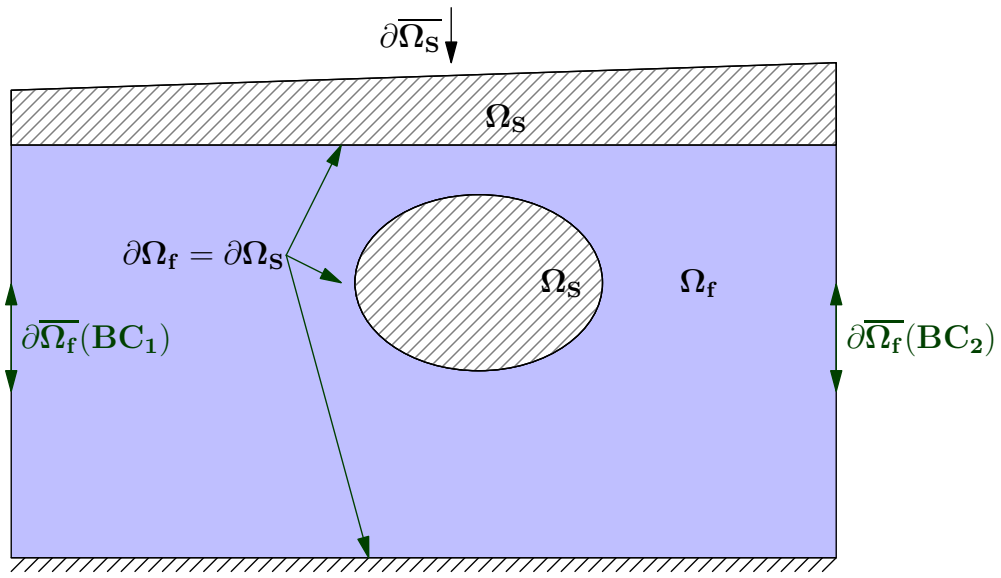


Figure 1.1: Computational domain showing respective fluid and solid subdomains, Ω_f and Ω_s. The shared boundaries are denoted $\partial\Omega_f = \partial\Omega_s$ and the solid boundary which is not shared by fluid is $\overline{\partial\Omega_s}$, while the fluid boundary not shared by solid $\overline{\partial\Omega_f}$.

- fast matrix free operator evaluation with minimal storage
- matrix operations are implemented in assembler code $M \times M$ routines to speed up computations
- the parallelism is "under the hood" demanding from the user only care in handling local versus global operations and array sizes
- by testing at the beginning of each run which one of the 3 readily implemented parallel algorithms behaves optimally it can be stated that the parallelism of Nek5000 is automatically tuned to each machine
- direct access to parameters at runtime
- geometry and boundary conditions exposed to the user via the .rea file
- handling complex geometries that can be imported from external codes

Nek5000 solves the unsteady incompressible two-dimensional, axisymmetric, or three-dimensional Stokes or Navier-Stokes equations with forced or natural convection heat transfer in both stationary (fixed) or time-dependent geometry. It also solves the compressible Navier-Stokes in the Low Mach regime, the magnetohydrodynamic equation (MHD). The solution variables are the fluid velocity $\mathbf{u} = (u_x, u_y, u_z)$, the pressure p, the temperature T. All of the above field variables are functions of space $\mathbf{x} = (x, y, z)$ and time t in domains Ω_f and/or Ω_s defined in Fig. 1.1. Additionally Nek5000 can handle conjugate heat transfer problems.

1.1 Incompressible Navier–Stokes equations

The governing equations of flow motion in dimensional form are

$$\rho(\partial_t \mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u}) = -\nabla p + \nabla\cdot\tau + \rho\mathbf{f} \ \text{, in } \Omega_f, \quad \text{(Momentum)} \tag{1.1}$$

where $\tau = \mu[\nabla\mathbf{u} + \nabla\mathbf{u}^T]$.

$$\nabla\cdot\mathbf{u} = 0 \ \text{, in } \Omega_f, \quad \text{(Continuity)} \tag{1.2}$$

If the fluid viscosity is constant in the entire domain the viscous stress tensor can be contracted $\nabla\cdot\tau = \mu\Delta\mathbf{u}$, therefore one may solve the Navier–Stokes equations in either the stress formulation, or no stress

- Variable viscosity requires the full stress tensor $\nabla\cdot\tau = \nabla\cdot\mu[\nabla\mathbf{u} + \nabla\mathbf{u}^T]$, and we shall refer to this as the stress formulation
- Constant viscosity leads to a simpler stress tensor $\nabla\cdot\tau = \mu\Delta\mathbf{u}$, which we refer to as the 'no stress' formulation

1.2 Non-dimensional Navier-Stokes

Let us introduce the following non-dimensional variables $\mathbf{x}^* = \frac{\mathbf{x}}{L}$, $\mathbf{u}^* = \frac{u}{U}$, $t^* = \frac{t}{L/U}$. For the pressure scale we have two options

- convective effects are dominant i.e. high velocity flows $p^* = \frac{p}{\rho U^2}$
- viscous effects are dominant i.e. creeping flows (Stokes flow) $p^* = \frac{pL}{\mu U}$

For highly convective flows we choose the first scaling of the pressure and obtain the non-dimensional Navier-Stokes:

$$\frac{\partial\mathbf{u}^*}{\partial t^*} + \mathbf{u}^*\cdot\nabla\mathbf{u}^* = -\nabla p^* + \frac{1}{Re}\nabla\cdot\tau^* + \frac{1}{Fr}\frac{\mathbf{f}}{g}. \tag{1.3}$$

where $\tau^* = [\nabla\mathbf{u}^* + \nabla\mathbf{u}^{*T}]$. The two non-dimensional numbers here are the Reynolds number $Re = \frac{\nu}{UL}$ Fr and the Froude number, defined as $Fr = \frac{U^2}{gL}$.

1.3 Energy equation

In addition to the fluid flow, Nek5000 computes automatically the energy equation

$$\rho c_p(\partial_t T + \mathbf{u}\cdot\nabla T) = \nabla\cdot(k\nabla T) + q_{vol} \ \text{, in } \Omega_f\cup\Omega_s \ \text{(Energy)} \tag{1.4}$$

1.4 Non-dimensional energy/passive scalar equation

A similar non-dimensionalization as for the flow equations using the non-dimensional variables $\mathbf{x}^* = \frac{\mathbf{x}}{L}$, $\mathbf{u}^* = \frac{u}{U}$, $t^* = \frac{t}{L/U}$, $T = \frac{T^*-T_0}{\delta T}$ leads to

$$\partial_{t^*} T^* + \mathbf{u}^*\cdot\nabla T^* = \frac{1}{Pe}\nabla\cdot\nabla T^* + q_{vol} \ \text{, in } \Omega_f\cup\Omega_s \ \text{(Energy)} \tag{1.5}$$

where $Pe = LU/\alpha$, with $\alpha = k/\rho c_p$.

1.5 Passive scalars

We can additionally solve a convection-diffusion equation for each passive scalar ϕ_i, i=1,2,... in $\Omega_f \cup \Omega_s$

$$(\rho c_p)_i(\partial_t \phi_i + \mathbf{u} \cdot \nabla \phi_i) = \nabla \cdot (k_i \nabla \phi_i) + (q_{vol})_i. \tag{1.6}$$

The terminology and restrictions of the temperature equations are retained for the passive scalars, so that it is the responsibility of the user to convert the notation of the passive scalar parameters to their thermal analogues. For example, in the context of mass transfer, the user should recognize that the values specified for temperature and heat flux will represent concentration and mass flux, respectively. Any combination of these equation characteristics is permissible with the following restrictions. First, the equation must be set to unsteady if it is time-dependent or if there is any type of advection. For these cases, the steady-state (if it exists) is found as stable evolution of the initial-value-problem. Secondly, the stress formulation must be selected if the geometry is time-dependent. In addition, stress formulation must be employed if there are traction boundary conditions applied on any fluid boundary, or if any mixed velocity/traction boundaries, such as symmetry and outflow/n, are not aligned with either one of the Cartesian x, y or z axes. Other capabilities of Nek5000 are the linearized Navier-Stokes for flow stability, magnetohydrodynamic flows etc.

1.6 Unsteady Stokes

In the case of flows dominated by viscous effects Nek5000 can solve the reduced Stokes equations

$$\rho(\partial_t \mathbf{u}) = -\nabla p + \nabla \cdot \tau + \rho \mathbf{f} \ \text{, in } \Omega_f \text{ (Momentum)} \tag{1.7}$$

where $\nabla \cdot \tau = \nabla \cdot \mu[\nabla \mathbf{u} + \nabla \mathbf{u}^T]$ and

$$\nabla \cdot \mathbf{u} = 0 \ \text{, in } \Omega_f \text{ (Continuity)} \tag{1.8}$$

Also here we can distinguish between the stress and non-stress formulation according to whether the viscosity is variable or not. The non-dimensional form of these equations can be obtained using the viscous scaling of the pressure.

1.7 Steady Stokes

If there is no time-dependence, then Nek5000 can further reduce to

$$-\nabla p + \nabla \cdot \tau + \rho \mathbf{f} = 0 \ \text{, in } \Omega_f \text{ (Momentum)} \tag{1.9}$$

where $\nabla \cdot \tau = \nabla \cdot \mu[\nabla \mathbf{u} + \nabla \mathbf{u}^T]$ and

$$\nabla \cdot \mathbf{u} = 0 \ \text{, in } \Omega_f \text{ (Continuity)} \tag{1.10}$$

1.8 Linearized Equations

In addition to the basic evolution equations described above, Nek5000 provides support for the evolution of small perturbations about a base state by solving the *linearized equations*

$$\rho(\partial_t \mathbf{u}_i{}' + \mathbf{u}\cdot\nabla\mathbf{u}_i{}' + \mathbf{u}_i'\cdot\nabla\mathbf{u}) = -\nabla p_i' + \mu\nabla^2\mathbf{u}_i', \qquad \nabla\cdot\mathbf{u}_i' = 0, \tag{1.11}$$

for multiple perturbation fields $i = 1, 2, \ldots$ subject to different initial conditions and (typically) homogeneous boundary conditions. These solutions can be evolved concurrently with the base fields $(\mathbf{u}, p, T)$. There is also support for computing perturbation solutions to the Boussinesq equations for natural convection. Calculations such as these can be used to estimate Lyapunov exponents of chaotic flows, etc.

1.9 Steady conduction

The energy Eq. 1.4 in which the advection term $\mathbf{u}\cdot\nabla T$ and the transient term $\partial_t T$ are zero. In essence this represents a Poisson equation.

1.10 Low-Mach Navier-Stokes

The compressible Navier-Stokes differ mathematically from the incompressible ones mainly in the divergence constraint $\nabla\cdot\mathbf{u} \neq 0$. In this case the system of equations is not closed and an additional equation of state (EOS) is required to connect the state variables, e.g. $p = f(\rho, T)$. However Nek5000 can only solve the Low Mach approximation of the compressible Navier-Stokes. The Low-Mach approximation decouples the pressure from the velocity leading to a system of equations which can be solved numerically in a similar fashion as the incompressible Navier-Stokes.

The Low Mach equations in non-dimensional form are

$$\begin{aligned}
&\rho\left(\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \nabla\cdot\boldsymbol{\tau} + \rho\mathbf{f} \qquad (1.12)\\
&\frac{\mathrm{d}\rho}{\mathrm{d}t} + \mathbf{u}\cdot\nabla\rho = -\rho\nabla\cdot\mathbf{u}\\
&\rho\left(\frac{\mathrm{d}T}{\mathrm{d}t} + \mathbf{u}\cdot\nabla T\right) = -\nabla\cdot k\nabla T
\end{aligned}$$

where $\boldsymbol{\tau} = \mu[\nabla\mathbf{u} + \nabla\mathbf{u}^T - \frac{2}{3}\nabla\cdot\mathbf{u}\mathbf{I}]$.

The implementation of the equation if state for the Low Mach formulation is for the moment hard-coded to be the ideal gas equation of state $p = \rho RT$. This allows for both variable density and variable viscosity. The system is solved by substituting $\rho = f(p, T)$ into the continuity equation and obtaining a so-called thermal divergence (the term $\nabla\cdot\mathbf{u}$ is given as a function of the temperature). A more detailed description on how these equations connect is given in section 1.10 as well as in the developer's manual.

1.11 Incompressible MHD equations

Magnetohydrodynamics is based on the idea that magnetic fields can induce currents in a moving conductive fluid, which in turn creates forces on the fluid and changing the magnetic field itself. The set of equations which describe MHD are a combination of the Navier-Stokes equations of fluid dynamics and Maxwell's equations of electromagnetism. These differential equations have to be solved simultaneously, and Nek5000 has an implementation for the incompressible MHD.

Consider a fluid of velocity $\mathbf{u}$ subject to a magnetic field $\mathbf{B}$ then the incompressible MHD equations are

$$\begin{aligned} \rho(\partial_t \mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u}) &= -\nabla p + \mu\Delta\mathbf{u} + \mathbf{B}\cdot\nabla\mathbf{B}\ , && (1.13)\\ \nabla\cdot\mathbf{u} &= 0 && (1.14)\\ \partial_t \mathbf{B} + \mathbf{u}\cdot\nabla\mathbf{B} &= -\nabla q + \eta\Delta\mathbf{B} + \mathbf{B}\cdot\nabla\mathbf{u}\ , \\ \nabla\cdot\mathbf{B} &= 0 \end{aligned}$$

where ρ is the density μ the viscosity, η resistivity, and pressure p.

The total magnetic field can be split into two parts: $\mathbf{B} = \mathbf{B_0} + \mathbf{b}$ (mean + fluctuations). The above equations become in terms of Elsässer variables ($\mathbf{z}^{\pm} = \mathbf{u} \pm \mathbf{b}$)

$$\frac{\partial \mathbf{z}^{\pm}}{\partial t} \mp (\mathbf{B}_0\cdot\nabla)\,\mathbf{z}^{\pm} + \left(\mathbf{z}^{\mp}\cdot\nabla\right)\mathbf{z}^{\pm} = -\nabla p + \nu_+\nabla^2\mathbf{z}^{\pm} + \nu_-\nabla^2\mathbf{z}^{\mp} \tag{1.15}$$

where $\nu_{\pm} = \nu \pm \eta$.

The important non-dimensional parameters for MHD are $Re = UL/\nu$ and the magnetic Re $Re_M = UL/\eta$.

1.12 Adaptive Lagrangian-Eulerian (ALE)

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

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= -\nabla p + \frac{1}{Re}\nabla\cdot(\nabla + \nabla^T)\mathbf{u} + NL, && (1.16)\\ \nabla\cdot\mathbf{u} &= 0 && (1.17) \end{aligned}$$

Here, NL represents the quadratic nonlinearities from the convective term.

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 (1.16): *Find* $(\mathbf{u},p)\in X^N\times Y^N$ *such that:*

$$\frac{d}{dt}(\mathbf{v},\mathbf{u}) = (\nabla\cdot\mathbf{v},p) - \frac{2}{Re}(\nabla\mathbf{v},\mathbf{S}) + (\mathbf{v},NL) + c(\mathbf{v},\mathbf{w},\mathbf{u}), \qquad (\nabla\cdot\mathbf{u},q) = 0, \tag{1.18}$$

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 (1.18) is the trilinear form involving the mesh velocity

$$c(\mathbf{v},\mathbf{w},\mathbf{u}) := \int_{\Omega(t)}\sum_{i=1}^{3}\sum_{j=1}^{3} v_i\frac{\partial w_j u_i}{\partial x_j}\,dV, \tag{1.19}$$

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 (1.18) 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 (1.18) is

$$\frac{1}{2\Delta t}\left[3(\mathbf{v}^n,\mathbf{u}^n)^n - 4(\mathbf{v}^{n-1},\mathbf{u}^{n-1})^{n-1} + (\mathbf{v}^{n-2},\mathbf{u}^{n-2})^{n-2}\right] = L^n(\mathbf{u}) + 2\widetilde{NL}^{n-1} - \widetilde{NL}^{n-2}. \tag{1.20}$$

Here, $L^n(\mathbf{u})$ accounts for all *linear* terms in (1.18), 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 (1.19), 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.

Chapter 2

Quick start

2.1 Download and Build

This chapter provides a quick overview to using Nek5000 for some basic flow problems provided in the `.../examples` directory.

Nek5000 runs under Linux or any Linux-like OS such as MAC, AIX, BG, Cray etc. The source is maintained in an svn repository and can be downloaded from the Nek5000 homepage (google nek5000) or, on linux systems, with the svn checkout command:

```
svn co https://svn.mcs.anl.gov/repos/nek5 nek5_svn
```

After downloading, build the tools by typing

```
cd nek5_svn/trunk/tools
maketools all
```

which will put the tools `genbox, genmap, n2to3, postx, prex,` and `pretex` in the top-level `/bin` directory (and will create `/bin` if it does not exist). It may be necessary to edit the `maketools` file to change the compilers (e.g., to pgf77/pgcc or ifort/icc). However, the default gfortran/gcc is generally fine.

In addition to the compiled tools, there are numerous scripts in

```
nek5_svn/trunk/tools/scripts
```

that are useful to have in the execution path, achieved either by adding this directory to the path or copying its contents to the top-level /bin directory. In the following, we assume that scripts such as `nek` and `nekb` are in the path. We further assume that the `nek5_svn/` prefix is implied in any future directory reference, unless otherwise specified.

2.2 A worked example

As a first example, we consider the eddy problem due to Walsh [1]. To get started, execute the following commands,

[1] O. Walsh, "Eddy solutions of the Navier-Stokes equations," *The NSE II-Theory and Numerical Methods*, J.G. Heywood, K. Masuda, R. Rautmann, and V.A. Solonikkov, eds., Springer, pp. 306–309 (1992)

```
cd
mkdir eddy
cd eddy
cp ~/nek5_svn/examples/eddy/* .
cp ~/nek5_svn/trunk/nek/makenek .
```

Modify `makenek`.

If you do not have `mpi` installed on your system, edit `makenek`, uncomment the `IFMPI="false"` flag, and change the Fortran and C compilers according to what is available on your machine. (Most any Fortran compiler save g77 or g95 will work.)

Nek5000 is written in F77 which has implicit typesetting as default. This means in practice that if the user defines a new variable in the user file and forgets to define its type explicitly then variable beginning with a character from I to N, its type is `INTEGER`. Otherwise, it is `REAL`.

This common type of mistake for a beginner can be avoided using a warning flag `-Wimplicit`. This flag warns whenever a variable, array, or function is implicitly declared. Has an effect similar to using the IMPLICIT NONE statement in every program unit.

Another useful flag may `-mcmodel` which allows for arrays of size larger than 2GB. This option tells the compiler to use a specific memory model to generate code and store data. It can affect code size and performance. If your program has global and static data with a total size smaller than 2GB, `-mcmodel=small` is sufficient. Global and static data larger than 2GB requires `-mcmodel=medium` or `-mcmodel=large`.

If you have `mpi` installed on your system or have made the prescribed changes to makenek, the eddy problem can be compiled as follows

Compiling nek. `makenek eddy_uv`

If all works properly, upon comilation the executable `nek5000` will be generated using `eddy_uv.usr` to provide user-supplied initial conditions and analysis. Note that if you encountered a problem during a prior attempt to build the code you should type

```
makenek clean;
makenek eddy_uv
```

Once compilation is successful, start the simulation by typing

Running a case: `nekb eddy_uv`

which runs the executable in the background (`nekb`, as opposed to `nek`, which will run in the foreground). If you are running on a multi-processor machine that supports MPI, you can also run this case via

A parallel run: `nekbmpi eddy_uv 4`

which would run on 4 processors. If you are running on a system that supports queuing for batch jobs (e.g., pbs), then the following would be a typical job submission command

```
nekpbs eddy_uv 4
```

In most cases, however, the details of the nekpbs script would need to be modified to accommodate an individual's user account, the desired runtime and perhaps the particular queue. Note that the scripts `nek, nekb, nekmpi, nekbmpi,` etc. perform some essential file manipulations prior to executing `nek5000`, so it is important to use them rather than invoking `nek5000` directly.

To check the error for this case, type

```
grep -i err eddy_uv.log | tail
```

or equivalently

```
grep -i err logfile | tail
```

where, because of the `nekb` script, `logfile` is linked to the `.log` file of the given simulation. If the run has completed, the above `grep` command should yield lines like

```
 1000  1.000000E-01  6.759103E-05  2.764445E+00  2.764444E+00  1.000000E+00  X err
 1000  1.000000E-01  7.842019E-05  1.818632E+00  1.818628E+00  3.000000E-01  Y err
```

which gives for the x- and y-velocity components the step number, the physical time, the maxiumum error, the maximum exact and computed values and the mean (bulk) values.

A common command to check on the progress of a simulation is

```
grep tep logfile | tail
```

which typically produces lines such as

```
Step    996, t= 9.9600000E-02, DT= 1.0000000E-04, C=  0.015 4.6555E+01 3.7611E-02
```

indicating, respectively, the step number, the physical time, the timestep size, the Courant (or CFL) number, the cumulative wall clock time (in seconds) and the wall-clock time for the most recent step. Generally, one would adjust Δt to have a CFL of $\sim$0.5.

2.3 Viewing the First 2D Example

The preferred mode for data visualization and analysis with Nek5000 is to use VisIt. For a quick peek at the data, however, we list a few commands for the native Nek5000 postprocessor. Assuming that the `maketools` script has been executed and that `/bin` is in the execution path, then typing
`postx`
in the working directory should open a new window with a sidebar menu. With the cursor focus in this window (move the cursor to the window and left click), hit `return` on the keyboard accept the default session name and click PLOT with the left mouse button. This should bring up a color plot of the pressure distribution for the first output file from the simulation (here, `eddy_uv.fld01`), which contains the geometry, velocity, and pressure.

Alternatively one can use the script *visnek*, to be found in `/scripts`. It is sufficent to run
`visnek eddy_uv` *(or the name of your session)*

to obatain a file named `eddy_uv.nek5000` which can be recognized in VisIt [2]

Plotting the error: For this case, the error has been written to `eddy_uv.fld11` by making a call to `outpost()` in the `userchk()` routine in `eddy_uv.usr`. The error in the velocity components is stored in the velocity-field locations and can be viewed with postx, or VisIt as before.

2.3.1 Modifying the First Example

A common step in the Nek5000 workflow is to rerun with a higher polynomial order. Typically, one runs a relatively low-order case (e.g., `lx1`=5) for one or two flow-through times and then uses the result as an initial condition for a higher-order run (e.g., `lx1`=8). We illustrate the procedure with the `eddy_uv` example.

Assuming that the contents of `nek5_svn/trunk/tools/scripts` are in the execution path, begin by typing

[2] https://wci.llnl.gov/simulation/computer-codes/visit/

```
cp eddy_uv eddy_new
```

which will copy the requisite `eddy_uv` case files to `eddy_new`. Next, edit `SIZE` and change the two lines defining `lx1` and `lxd` from

```
      parameter (lx1=8,ly1=lx1,lz1=1,lelt=300,lelv=lelt)
      parameter (lxd=12,lyd=lxd,lzd=1)
```

to

```
      parameter (lx1=12,ly1=lx1,lz1=1,lelt=300,lelv=lelt)
      parameter (lxd=18,lyd=lxd,lzd=1)
```

Then recompile the source by typing

```
makenek eddy_new
```

Next, edit `eddy_new.rea` and change the line

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

(found roughly 33 lines from the bottom of the file) to

```
           1 PRESOLVE/RESTART OPTIONS  *****
eddy_uv.fld12
```

which tells nek5000 to use the contents of `eddy_uv.fld12` as the initial condition for `eddy_new`. The simulation is started in the usual way:

```
nekb eddy_new
```

after which the command

```
grep err logfile | tail
```

will show a much smaller error ($\sim 10^{-9}$) than the `lx1=8` case.

Note that one normally would not use a restart file for the *eddy* problem, which is really designed as a convergence study. The purpose here, however, was two-fold, namely, to illustrate a change of order and its impact on the error, and to demonstrate the frequently-used restart procedure. However for a higher order timestepping scheme an accurate restart would require a number of field files of the same size (+1) as the order of the multistep scheme

Chapter 3

User files

3.1 Case set-up .usr

Each simulation is defined by three files, the .rea file, the .usr file, and the SIZE file. In addition, there is a derived .map file that is generated from the .rea file by running `genmap` which will determine how the elements will be split across processors in the case of a parallel run. SIZE controls (at compile time) the polynomial degree used in the simulation, as well as the space dimension $d = 2$ or 3.

The SESSION.NAME file contains the current run, it must provide the name of the .rea file and the path to it. It does not however need to correspond to an .usr file of an identical name. This allows for different test cases (.usr files) that use the same geometry and boundary conditions (.rea files).

This chapter provides an introduction to the basic files required to set up a Nek5000 simulation.

3.1.1 Contents of .usr file

The most important interface to Nek5000 is the set of Fortran subroutines that are contained in the `.usr` file. This file allows direct access to all runtime variables. Here, the user may specify spatially varying properties (e.g., viscosity), volumetric heating sources, body forces, and so forth. One can also 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. The `.usr` files provided in the `/examples/...` directories illustrate several of the more common analysis tools. For instance, 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 routines for computing the vorticity or the scalar λ_2 for vortex identification, and so forth.

Routines in .usr file

The routine `uservp` specifies the variable properties of the governing equations. This routine is called once per processor, and once per discrete point therein.

Equation	udiff	utrans	ifield
Momentum Eq.1.1	ρ	μ	1
Energy Eq.1.4	ρc_p	k	2
Passive scalar Eq.1.6	$(\rho c_p)_i$	k_i	i-1

```
      subroutine uservp (ix,iy,iz,eg)
      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'

      integer iel
      iel = gllel(eg)

      udiff =0.
      utrans=0.

      return
      end
```

The routine `userdat` is called right after the geometry is loaded into NEK5000 and prior to the distribution of the GLL points. This routine is called once per processor but for all the data on that processor. At this stage the elements can be modified as long as the topology is preserved. It is also possible to alter the type of boundary condition that is initially attributed in the `.rea` file, as illustrated below (the array `cbc(face,iel,field)` contains the boundary conditions per face and field of each element). Note the spacing allocated to each BC string is of three units.

```
      subroutine usrdat
      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'
      integer iel,f

      do iel=1,nelt  !  Force flux BCs
      do f=1,2*ndim
         if (cbc(f,iel,1).eq.'W  ') cbc(f,iel,2) = 'f  ' ! flux BC for temperature
      enddo
      enddo

      return
      end
```

The routine `usrdat2` is called after the GLL points were distributed and allows at this point only for affine transformations of the geometry.

```
      subroutine usrdat2
      include 'SIZE'
      include 'TOTAL'

      return
      end
```

The routine `userf` is called once for each point and provides the force term in Eq.1.1. Not that according to the dimensionalization in Eq.1.1 the force term $\mathbf{f}$ is in fact multiplied by the density ρ.

```
      subroutine userf  (ix,iy,iz,eg)
      include 'SIZE'
```

```
      include 'TOTAL'
      include 'NEKUSE'

      ffx = 0.0
      ffy = 0.0
      ffz = 0.0

      return
      end
```

Similarly to `userf` the routine `userq` provides the force term in Eq.1.4 and the subsequent passive scalar equations according to Eq.1.6.

```
      subroutine userq  (ix,iy,iz,eg)
      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'

      qvol   = 0.

      return
      end
```

The boundary conditions are assigned in `userbc` for both the fluid, temperature and all other scalars. An extensive list of such possible boundary conditions is available in Section. 4.4.1.

```
      subroutine userbc (ix,iy,iz,iside,ieg)
      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'

      ux=0.0
      uy=0.0
      uz=0.0
      temp=0.0
      flux = 1.0

      return
      end
```

Initial conditions are attributed in `useric` similarly to the boundary conditions

```
      subroutine useric (ix,iy,iz,ieg)
      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'

      uy=0.0
      ux=0.0
      uz=1.0

      return
      end
```

The routine `userchk` is called once per processor after each timestep (and once after the initialization is finished). This is the section where the solution can be interrogated and subsequent changes can be made.

Dimensional parameters	Non-dimensional parameters
p1=ρ	p1=1
p2=ν	p2=1/Re (-Re)
p7=ρC_p	p7=1
p8=k	p8=1/Pe (-Pe)

```
      subroutine userchk
      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'

      call outpost(vx,vy,vz,pr,t,'ext')

      return
      end
```

The routine `usrdat3` is not widely used, however it shares the same properties with `usrdat2`.

```
      subroutine usrdat3
      include 'SIZE'
      include 'TOTAL'
c
      return
      end
```

Nek5000 can solve the dimensional or non-dimensional equations by setting the following parameters

alternatively the variable properties can be set in the USERVP routine.

What is a SESSION file?

To run NEK5000, each simulation must have a SESSION.NAME file. This file is read in by the code and gives the path to the relevant files describing the structure and parameters of the simulation. The SESSION.NAME file is a file that contains the name of the simulation and the full path to supporting files. For example, to run the eddy example from the repository, the SESSION.NAME file would look like:

```
eddy_uv\\
/homes/user\_ name/nek5\_ svn/examples/eddy/
```

3.2 Problem size file SIZE

The SIZE file defines the problem size, i.e. spatial points at which the solution is to be evaluated within each element, number of elements per processor etc. The SIZE 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 SIZE 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 determines the *maximum* number of elements *per processor*.

The total size of the problem is `lx1*ly1*lz1*lelt`.

3.2.1 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 note 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 case so that the executable can be run without recompiling on any admissible number of processors ($P_{\text{mem}} \leq P \leq E$, where P_{mem} is the value computed above).

3.3 Geometry and Parameters file .rea

The `.rea` file consists of several sections:

Parameters and logical switches

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.

passive scalar data This information can be specified also in the `.uservp` routine in the .usr file. If specified in the .rea file then the coefficients for the conductivity term are listed in ascending order for passive scalars ranging `1..9` followed by the values for the ρc_p coefficients.

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

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

Next we have the logical switches as follow, a detailed explanation to be found in Sec:7.1.4

```
         13  LOGICAL SWITCHES FOLLOW
T     IFFLOW
T     IFHEAT
T     IFTRAN
T T F F F F F F F F IFNAV & IFADVC (convection in P.S. fields)
F F T T T T T T T T T IFTMSH (IF mesh for this field is T mesh)
F     IFAXIS
F     IFSTRS
F     IFSPLIT
F     IFMGRID
F     IFMODEL
F     IFKEPS
F     IFMVBD
F     IFCHAR
```

Mesh and boundary condition info

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. A line of the following type may be encountered at the beginning of the mesh section of the area file.

```
3.33333       3.33333      -0.833333      -1.16667      XFAC,YFAC,XZERO,YZERO
```

This part is to be read by PRENEK and provides the origin of the system of coordinates `XZERO;YZERO` as well as the size of the cartesian units `XFAC;YFAC`. This one line has no impact on the mesh as being read in NEK.

The header of the mesh data may have the following representation

```
 **MESH DATA** 6 lines are X,Y,Z;X,Y,Z. Columns corners 1-4;5-8
      226  3         192           NEL,NDIM,NELV
     ELEMENT           1 [    1A]    GROUP    0
```

The header states first how many elements are available in total (226), what dimension is the the problem (here three dimensional), and how many elements are in the flow mesh (192).

`Face {1,2,3,4}`				
$x_{1,\ldots,4} =$	0.000000E+00	0.171820E+00	0.146403E+00	0.000000E+00
$y_{1,\ldots,4} =$	0.190000E+00	0.168202E+00	0.343640E+00	0.380000E+00
$z_{1,\ldots,4} =$	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
`Face {5,6,7,8}`				
$x_{5,\ldots,8} =$	0.000000E+00	0.171820E+00	0.146403E+00	0.000000E+00
$y_{5,\ldots,8} =$	0.190000E+00	0.168202E+00	0.343640E+00	0.380000E+00
$z_{5,\ldots,8} =$	0.250000E+00	0.250000E+00	0.250000E+00	0.250000E+00

(a) Descriptor

Table 3.1: Geometry description in .rea file

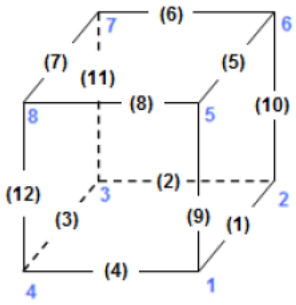


Figure 3.1: Geometry description in .rea file (sketch of one element ordering)

curvature This section describes the deformation for elements that are curved. Currently-supported curved side or edge definitions include "C" for circles, "s" for spheres, and "m" for midside-node positions associated with quadratic edge displacement. If no curved data is available the section remains empty. Example

The section header may look like this

```
640 Curved sides follow IEDGE,IEL,CURVE(I),I=1,5, CCURVE
```

and the data is stored as follows

IEDGE	IEL	CURVE(12,1,IEL)	CURVE(12,2..5,IEL)	CCURVE(12,IEL)
3	1	1.0000	0.0000	C
7	1	1.0000	0.0000	C

The array `CCURVE` (char curve) holds a character denoting the type of curved boundary, while the array `CURVE` holds the actual information about the curved boundary. There are up to five available components in the `CURVE` array in case more information is needed by other implementations, that do not represent the default. We may have

- 'C' stands for circle and is given by the radius of the circle, thus filling in only the first component of the `CURVE(12,1,NEL)`
- 'S' stands for sphere and is given by the radius and the center of the sphere, thus filling the first 4 components of the `CURVE(12,1...4,NEL)`

- 'M' is given by the coordinates of the midside-node, thus filling the first 3 components of the `CURVE(12,1...3,NEL)`, and leads to a second order reconstruction of the face.

Both 'C' and 'S' types allow for a surface of as high order as the polynomial used in the spectral method, since they have an underlying analytical description, any circle arc can be fully determined by the radius and end points. However for the 'M' curved element descriptor the surface can be reconstructed only up to second order. This can be later updated to match the high-order polynomial after the GLL points have been distributed across the boundaries. In .usrdat2 the user can move the geometry to match the intended surface, followed by a call to the subroutine 'fixgeom' which can realign the point distribution in the interior of the element.

boundary conditions Boundary conditions (BCs) are specified for each face of each element, for each `field` (velocity, temperature, passive scalar #1, etc.). A common BC is `P`, which indicates that an element face is connected to another element to establish a periodic BC. Many of the BCs 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, including free-surface, moving boundary, heat flux, convective cooling, etc.

The section header may look like this

```
***** FLUID BOUNDARY CONDITIONS *****
```

and the data is stored as follows

CBC	IEL	IEDGE	CONN-IEL	CONN-IEDGE	redundant
E	1	1	4.00000	3.00000	0.00000
..	..	..	..	..	..
W	5	3	0.00000	0.00000	0.00000
..	..	..	..	..	..
P	5	5	149.000	6.00000	0.00000

Output info

restart conditions

Here, one 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. One can also specify that, for example, the velocity is to come from one file and the temperature from another. The initial time is taken from the last specified restart file, but this can be overridden.

History points

The following section defines history points in the `.rea` file, see example `vortex/r1854a.rea`, or `shear4/shear4.rea`

```
0 PACKETS OF DATA FOLLOW\\
***** HISTORY AND INTEGRAL DATA *****\\
    56 POINTS. H code, I,J,H,IEL \\
```

```
UVWP    H     31     31    1    6\\
UVWP    H     31     31    31   6\\
UVWP    H     31     31    31   54\\
 "      "      "      "     "    "\\
```

The `"56 POINTS"` line needs to be followed by 56 lines of the type shown. However, in each of the following lines, which have the `UVWP` etc., location is CRUCIAL, it must be layed out exactly as indicated above[1], it is therefore advisable to refer to the examples `vortex, shear4`. 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`. [2]

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. The "H" indicates that the entry corresponds to a requested history point. A note of caution: if the `ijk` values (5 5 5 in the preceding example line) exceed `lx1,ly1,lz1` of your SIZE 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.

output specifications Outputs are discussed in a separate section below.

It is important to note that Nek5000 currently supports two input file 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 ($>$ 100000, say).

3.3.1 Parameters

- ρ, the density, is taken to be time-independent and constant; however, in a multi-fluid system different fluids can have different value of constant density.
- μ, the dynamic viscosity can vary arbitrarily in time and space; it can also be a function of temperature (if the energy equation is included) and strain rate invariants (if the stress formulation is selected).
- σ, the surface-tension coefficient can vary arbitrarily in time and space; it can also be a function of temperature and passive scalars.

[1] these lines contain character strings, they use formatted reads

[2] the indicated point would really be at the middle of the element only if lx1=9

- $\overline{\beta}$, the effective thermal expansion coefficient, is assumed time-independent and constant.
- $\mathbf{f}(t)$, the body force per unit mass term can vary with time, space, temperature and passive scalars.
- ρc_p, the volumetric specific heat, can vary arbitrarily with time, space and temperature.
- ρL, the volumetric latent heat of fusion at a front, is taken to be time-independent and constant; however, different constants can be assigned to different fronts.
- k, the thermal conductivity, can vary with time, space and temperature.
- q_{vol}, the volumetric heat generation, can vary with time, space and temperature.
- h_c, the convection heat transfer coefficient, can vary with time, space and temperature.
- h_{rad}, the Stefan-Boltzmann constant/view-factor product, can vary with time, space and temperature.
- T_∞, the environmental temperature, can vary with time and space.
- T_{melt}, the melting temperature at a front, is taken with time and space; however, different melting temperature can be assigned to different fronts.

In the solution of the governing equations together with the boundary and initial conditions, Nek5000 treats the above parameters as pure numerical values; their physical significance depends on the user's choice of units. The system of units used is arbitrary (MKS, English, CGS, etc.). However, the system chosen must be used consistently throughout. For instance, if the equations and geometry have been non-dimensionalized, the μ/ρ in the fluid momentum equation is in fact the inverse Reynolds number, whereas if the equations are dimensional, μ/ρ represents the kinematic viscosity with dimensions of $length^2/time$.

3.4 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.

Chapter 4

Geometry

Note that in case of any changes in the SIZE file, a recompilation is necessary.

4.1 Setting up the geometry

4.1.1 Rectangular geometries

4.1.2 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. 4.1. 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 box immediately behind the
#                refined cylinder in Ugo's cyl+b.l. run.
#
#
#=========================================================
#
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.

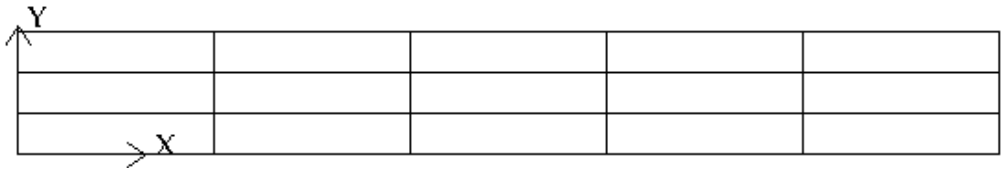


Figure 4.1: Axisymmteric pipe mesh

- 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 distribution 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 distribution 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 $\mathbf{u} = 0$.

4.1.3 Graded Mesh

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

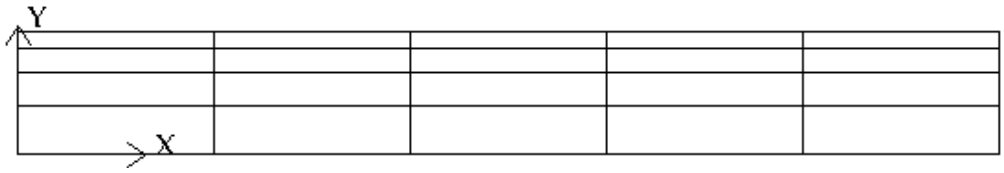


Figure 4.2: Axisymmteric pipe mesh, graded

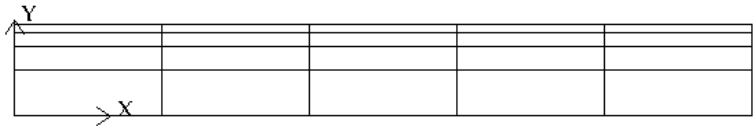


Figure 4.3: Axisymmteric pipe mesh, user specified

```
-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. 4.2.

4.1.4 User-Specified Distribution

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. 4.3.

4.1.5 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 through the usrdat2 routine provided in the .usr file. The routine usrdat2 is called by nek5000 immediately after the geometry, as specified 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

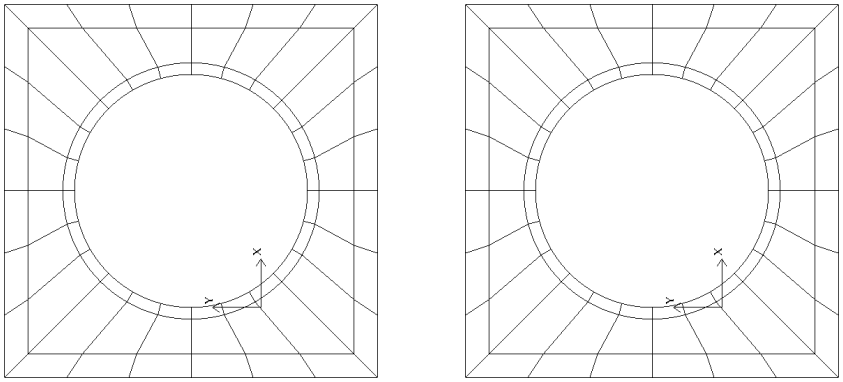
```
subroutine usrdat2
include 'SIZE'
include 'TOTAL'

real lambda

ntot = nx1*ny1*nz1*nelt

lambda = 3.
A      = 0.1

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

param(59) = 1.  ! Force nek5 to recognize element deformation.

return
end
```

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

Figure 4.4: Axisymmteric pipe mesh



(a) Annuli mesh (b) Annuli mesh

Figure 4.5: Cylinder mesh

```
***** 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. 4.4.

4.1.6 Cylindrical/Cartesian-transition Annuli

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 nek5000/doc files *box7.2d* and *box7.3d*. Figure 4.5a 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 is shown in Fig. 4.5a. 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.

4.2 Extrusion/Mirroring

4.2.1 Building Extruded Meshes with n2to3

In nek5000/tools, there is a code n2to3.f that can be compiled with your local fortran compiler (preferably 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. Assuming you have a valid two-dimensional mesh, n2to3 is straightforward to run. Below is a typical session, upon typing `n2to3` the user is prompted at the command line

```
 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,0):
v
 Enter Z (6) boundary condition (v,0):
0
```

```
 this is cbz: v  0   <---

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

In this context CEM stands for computational electromagnetics, a spin-off of the original Nek5000 code.

The domain in which the fluid flow/heat transfer problem is solved consists of two distinct subdomains. The first subdomain is that part of the region occupied by fluid, denoted Ω_f, while the second subdomain is that part of the region occupied by a solid, denoted Ω_s. These two subdomains are depicted in Fig. 1.1. The entire domain is denoted as $D = \Omega_f \cup \Omega_s$. The fluid problem is solved in the domain Ω_f, while the temperature in the energy equation is solved in the entire domain; the passive scalars can be solved in either the fluid or the entire domain.

We denote the entire boundary of Ω_f as $\partial\Omega_f$, that part of the boundary of Ω_f which is not shared by Ω_s as $\overline{\partial\Omega_f}$, and that part of the boundary of Ω_f which is shared by Ω_s. In addition, $\partial\Omega_s, \overline{\partial\Omega_s}$ are analogously defined. These distinct portions of the domain boundary are illustrated in Fig.1.1. The restrictions on the domain for Nek5000 are itemized below.

- The domain $\Omega = \Omega_f \cup \Omega_s$ must correspond either to a planar (Cartesian) two-dimensional geometry, or to the cross-section of an axisymmetric region specified by revolution of the cross-section about a specified axis, or by a (Cartesian) three-dimensional geometry.

- For two-dimensional and axisymmetric geometries, the boundaries of both subdomains, $\partial\Omega_f$ and $\partial\Omega_s$, must be representable as (or at least approximated by) the union of straight line segments, splines, or circular arcs.

- Nek5000 can interpret a two-dimensional image as either a planar Cartesian geometry, or the cross-section of an axisymmetric body. In the case of the latter, it is assumed that the y-direction is the radial direction, that is, the axis of revolution is at y=0. Although an axisymmetric geometry is, in fact, three-dimensional, Nek5000 can assume that the field variables are also axisymmetric (that is, do not depend on azimuth, but only y, that is, radius, x, and t), thus reducing the relevant equations to "two-dimensional" form.

Fully general three-dimensional meshes generated by other softwares packages can be input to PRENEK as import meshes.

4.3 Moving Geometry

If the imposed boundary conditions allow for motion of the boundary during the solution period (for example, moving walls, free-surfaces, melting fronts, fluid layers), then the geometry of the computational domain is automatically considered in Nek5000 as being time-dependent.

For time-dependent geometry problems, a mesh velocity $\mathbf{w}$ is defined at each collocation point of the computational domain (mesh) to characterize the deformation of the mesh. In the solution of the mesh velocity, the value of the mesh velocity at the moving boundaries is first computed using appropriate kinematic conditions (for free-surfaces, moving walls and fluid layers) or dynamic conditions (for melting fronts). On all other external boundaries, the normal mesh velocity on the boundary is always set to zero. In the tangential direction, either a zero tangential velocity

condition or a zero tangential traction condition is imposed; this selection is automatically performed by Nek5000 based on the fluid and/or thermal boundary conditions specified on the boundary. However, under special circumstances the user may want to override the defaults set by Nek5000, this is described in the PRENEK manual in Section 5.7.[1] If the zero tangential mesh velocity is imposed, then the mesh is fixed in space; if the zero traction condition is imposed, then the mesh can slide along the tangential directions on the boundary. The resulting boundary-value-problem for the mesh velocity is solved in Nek5000 using a elastostatic solver, with the Poisson ratio typically set to zero. The new mesh geometry is then computed by integrating the mesh velocity explicitly in time and updating the nodal coordinates of the collocation points.

Note that the number of macro-elements, the order of the macro-elements and the topology of the mesh remain *unchanged* even though the geometry is time-dependent. The use of an arbitrary-Lagrangian-Eulerian description in Nek5000 ensures that the moving fronts are tracked with the minimum amount of mesh distortion; in addition, the elastostatic mesh solver can handle moderately large mesh distortion. However, it is the responsibility of the user to decide when a mesh would become "too deformed" and thus requires remeshing. The execution of the program will terminate when the mesh becomes unacceptable, that is, a one-to-one mapping between the physical coordinates and the isoparametric local coordinates for any macro-element no longer exists.

4.4 Boundary and initial conditions

4.4.1 Boundary Conditions

The boundary conditions for the governing equations given in the previous section are now described.

The boundary conditions can be imposed in various ways:

- when the mesh is generated with `genbox`, as will be explained in Section 4.1.6
- when the .rea file is read in PRENEK or directly in the .rea file
- directly in the .rea file
- in the subroutine `userbc`

The general convention for boundary conditions in the .rea file is

- upper case letters correspond to Primitive boundary conditions, as given in Table **??**
- lower case letters correspond to user defined boundary conditions, see Table **??**

Since there are no supporting tools that will correctly populate the .rea file with the appropriate values, temperature, velocity, and flux boundary conditions are typically lower case and values must be specified in the `userbc` subroutine in the .usr file.

4.4.2 Fluid Velocity

Two types of boundary conditions are applicable to the fluid velocity : essential (Dirichlet) boundary condition in which the velocity is specified; natural (Neumann) boundary condition in which the

[1] This manual is old may soon be deprecated

Identifier	Description	Parameters	No of Parameters
P	periodic	periodic element and face	2
V	Dirichlet velocity	u,v,w	3
O	outflow	-	0
W	wall (no slip)	-	0
F	flux	flux	1
SYM	symmetry	-	0
A	axisymmetric boundary	-	0
MS	moving boundary	-	0
ON	Outflow, Normal	-	0
E	Interior boundary	Neighbour element ID	2

Table 4.1: Primitive boundary conditions (flow velocity)

Identifier	Description
v	user defined Dirichlet velocity
t	user defined Dirichlet temperature
f	user defined flux

Table 4.2: User defined boundary conditions (flow velocity)

traction is specified. For segments that constitute the boundary $\partial\Omega_f$, see Fig. 1.1, one of these two types of boundary conditions must be assigned to each component of the fluid velocity. The fluid boundary condition can be *all Dirichlet* if all velocity components of $\mathbf{u}$ are specified; or it can be *all Neumann* if all traction components $\mathbf{t} = [-p\mathbf{I}+\mu(\nabla\mathbf{u}+(\nabla\mathbf{u})^T)]\cdot\mathbf{n}$, where $\mathbf{I}$ is the identity tensor, $\mathbf{n}$ is the unit normal and μ is the dynamic viscosity, are specified; or it can be *mixed Dirichlet/Neumann* if Dirichlet and Neumann conditions are selected for different velocity components. Examples for all Dirichlet, all Neumann and mixed Dirichhlet/Neumann boundaries are wall, free-surface and symmetry, respectively. If the nonstress formulation is selected, then traction is not defined on the boundary. In this case, any Neumann boundary condition imposed must be homogeneous; i.e., equal to zero. In addition, mixed Dirichlet/Neumann boundaries must be aligned with one of the Cartesian axes.

For flow geometry which consists of a periodic repetition of a particular geometric unit, the periodic boundary conditions can be imposed, as illustrated in Fig. 1.1.

The open(outflow) boundary condition ("O") arises as a natural boundary condition from the variational formulation of Navier Stokes. We identify two situations

- In the non-stress formulation, open boundary condition ('Do nothing')

$$[-p\mathbf{I} + \nu(\nabla\mathbf{u})] \cdot \mathbf{n} = 0 \tag{4.1}$$

- In the stress formulation, free traction boundary condition

$$[-p\mathbf{I} + \nu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)] \cdot \mathbf{n} = 0 \tag{4.2}$$

Identifier	Description	Parameters	No of Parameters
T	Dirichlet temperature/scalar	value	1
O	outflow	-	0
P	periodic boundary	-	0
I	insulated (zero flux) for temperature		0

Table 4.3: Primitive boundary conditions (Temperature and Passive scalars)

- the symmetric boundary condition ("SYM") is given as

$$\mathbf{u} \cdot \mathbf{n} = 0 \,, \tag{4.3}$$
$$(\nabla \mathbf{u} \cdot \mathbf{t}) \cdot \mathbf{n} = 0 \tag{4.4}$$

  where $\mathbf{n}$ is the normal vector and $\mathbf{t}$ the tangent vector. If the normal and tangent vector are not aligned with the mesh the stress formulation has to be used.

- the periodic boundary condition ("P") needs to be prescribed in the .rea file since it already assigns the last point to first via $\mathbf{u}(\mathbf{x}) = \mathbf{u}(\mathbf{x} + L)$, where L is the periodic length.

- the wall boundary condition ("W") corresponds to $\mathbf{u} = 0$.

For a fully-developed flow in such a configuration, one can effect great computational efficiencies by considering the problem in a single geometric unit (here taken to be of length L), and requiring periodicity of the field variables. Nek5000 requires that the pairs of sides (or faces, in the case of a three-dimensional mesh) identified as periodic be identical (i.e., that the geometry be periodic).

For an axisymmetric flow geometry, the axis boundary condition is provided for boundary segments that lie entirely on the axis of symmetry. This is essentially a symmetry (mixed Dirichlet/Neumann) boundary condition in which the normal velocity and the tangential traction are set to zero.

For free-surface boundary segments, the inhomogeneous traction boundary conditions involve both the surface tension coefficient σ and the mean curvature of the free surface.

Passive scalars and Temperature

The three types of boundary conditions applicable to the temperature are: essential (Dirichlet) boundary condition in which the temperature is specified; natural (Neumann) boundary condition in which the heat flux is specified; and mixed (Robin) boundary condition in which the heat flux is dependent on the temperature on the boundary. For segments that constitute the boundary $\partial\Omega'_f \cup \partial\Omega'_s$ (refer to Fig. 2.1), one of the above three types of boundary conditions must be assigned to the temperature.

The two types of Robin boundary condition for temperature are : convection boundary conditions for which the heat flux into the domain depends on the heat transfer coefficient h_c and the difference between the environmental temperature T_∞ and the surface temperature; and radiation boundary conditions for which the heat flux into the domain depends on the Stefan-Boltzmann constant/view-factor product h_{rad} and the difference between the fourth power of the environmental temperature T_∞ and the fourth power of the surface temperature.

Identifier	Description
t	user defined Dirichlet temperature
c	Newton cooling
f	user defined flux

Table 4.4: User defined boundary conditions (Temperature and Passive scalars)

- open boundary condition ("O")
$$k(\nabla T) \cdot \mathbf{n} = 0 \tag{4.5}$$
- insulated boundary condition ("I")
$$k(\nabla T) \cdot \mathbf{n} = 0 \tag{4.6}$$
where $\mathbf{n}$ is the normal vector and $\mathbf{t}$ the tangent vector. If the normal and tangent vector are not aligned with the mesh the stress formulation has to be used.
- the periodic boundary condition ("P") needs to be prescribed in the .rea file since it already assigns the last point to first via $\mathbf{u}(\mathbf{x}) = \mathbf{u}(\mathbf{x} + L)$, where L is the periodic length.
- Newton cooling boundary condition ("c")
$$k(\nabla T) \cdot \mathbf{n} = h(T - T_\infty) \tag{4.7}$$
- flux boundary condition ("f")
$$k(\nabla T) \cdot \mathbf{n} = f \tag{4.8}$$

The boundary conditions for the passive scalar fields are analogous to those used for the temperature field. Thus, the temperature boundary condition menu will reappear for each passive scalar field so that the user can specify an independent set of boundary conditions for each passive scalar field.

4.4.3 Internal Boundary Conditions

In the spatial discretization, the entire computational domain is subdivided into macro-elements, the boundary segments shared by any two of these macro-elements in Ω_f and Ω_s are denoted as internal boundaries. For fluid flow analysis with a single-fluid system or heat transfer analysis without change-of-phase, internal boundary conditions are irrelevant as the corresponding field variables on these segments are part of the solution. However, for a multi-fluid system and for heat transfer analysis with change-of-phase, special conditions are required at particular internal boundaries, as described in the following.

For a fluid system composes of multiple immiscible fluids, the boundary (and hence the identity) of each fluid must be tracked, and a jump in the normal traction exists at the fluid-fluid interface if the surface tension coefficient is nonzero. For this purpose, the interface between any two fluids of different identity must be defined as a special type of internal boundary, namely, a fluid layer; and the associated surface tension coefficient also needs to be specified.

In a heat transfer analysis with change-of-phase, Nek5000 assumes that both phases exist at the start of the solution, and that all solid-liquid interfaces are specified as special internal boundaries, namely, the melting fronts. If the fluid flow problem is considered, i.e., the energy equation is solved in conjunction with the momentum and continuity equations, then only the common boundary between the fluid and the solid (i.e., all or portion of $\partial\overline{\Omega}_f'$ in Fig. 1.1) can be defined as the melting front. In this case, segments on $\partial\overline{\Omega}_f'$ that belong to the dynamic melting/freezing interface need to be specified by the user. Nek5000 always assumes that the density of the two phases are the same (i.e., no Stefan flow); therefore at the melting front, the boundary condition for the fluid velocity is the same as that for a stationary wall, that is, all velocity components are zero. If no fluid flow is considered, i.e., only the energy equation is solved, then any internal boundary can be defined as a melting front. The temperature boundary condition at the melting front corresponds to a Dirichlet condition; that is, the entire segment maintains a constant temperature equal to the user-specified melting temperature T_{melt} throughout the solution. In addition, the volumetric latent heat of fusion ρL for the two phases, which is also assumed to be constant, should be specified.

4.4.4 Initial Conditions

For time-dependent problems Nek5000 allows the user to choose among the following types of initial conditions for the velocity, temperature and passive scalars:

- Zero initial conditions: default; if nothing is specified.
- Fortran function: This option allows the user to specify the initial condition as a fortran function, e.g., as a function of x, y and z.
- Presolv: For a temperature problem the presolv option gives the steady conduction solution as initial condition for the temperature. For a fluid problem this option *can* give the steady Stokes solution as the initial condition for the velocity *provided* that the classical splitting scheme is *not* used.
- Restart: this option allows the user to read in results from an earlier simulation, and use these as initial conditions.

A tabulated summary of the compatibility of these initial condition options with various other solution strategies/parameters is given in the appendix.

4.5 Mesh Partioning for Parallel Computing

Genmap is spectral graph partitioning tool, similar to e.g. METIS, which partitions the graph associated to the mesh to assure optimal communication time in HPC applications. Let us consider a simple mesh such as the one in Fig. 4.6. The vertices are distributed in a random fashion, which is the way they may be provided by some mesh generator. Let us assume the vertices are here given as

$V_1 = (-1, 0)$, $V_2 = (0, 1)$, $V_3 = (-1, 2)$, $V_4 = (-1, 1)$, $V_5 = (0, 2)$, $V_6 = (0, 0)$, $V_7 = (1, 1)$, $V_8 = (1, 0)$

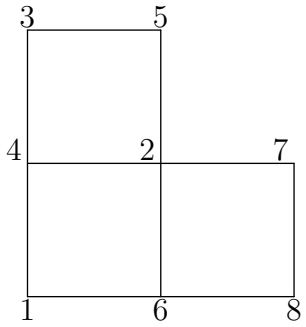


Figure 4.6: Two-dimensional mesh

The geometry is already stored in the .rea file by the point coordinates, and not vertex numbers

`Element` $1 = [V_1\ V_6\ V_2\ V_4]$

$x_{1,\dots,4} =$ -1. 0. 0. -1.

$y_{1,\dots,4} = 0.$ 0. 1. 1.

`Element` $2 = [V_8\ V_7\ V_2\ V_6]$

$x_{1,\dots,4} = 1.$ 1. 0. 0.

$y_{1,\dots,4} = 0.$ 1. 1. 0.

`Element` $3 = [V_5\ V_3\ V_4\ V_2]$

$x_{1,\dots,4} = 0.$ -1. -1. 0.

$y_{1,\dots,4} = 2\ .$ 2. 1. 1.

Let us a regard the mesh in Fig. 4.6 as a graph of N vertices and M edges, $G(V_N, E_M)$. We define the Laplacian matrix associated to a graph G as $L(G)$. We define as the degree of a node V_i the number of incident edges, e.g. in Fig. 4.6 $deg(V_2) = 4$ and $deg(V_6) = 3$.

$$L(G)_{ij} = \begin{cases} i = j & \text{de}g(V_i) \\ i \neq j & -1 \text{ if there is and edge (i,j) and 0 otherwise} \end{cases} \tag{4.9}$$

$$L(G) = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1| & 2 & 0 & 0 & -1 & 0 & -1 & 0 & 0 \\ 2| & 0 & 4 & 0 & -1 & -1 & -1 & -1 & 0 \\ 3| & 0 & 0 & 2 & -1 & -1 & 0 & 0 & 0 \\ 4| & -1 & -1 & -1 & 3 & 0 & 0 & 0 & 0 \\ 5| & 0 & -1 & -1 & 0 & 2 & 0 & 0 & 0 \\ 6| & -1 & -1 & 0 & 0 & 0 & 3 & 0 & -1 \\ 7| & 0 & -1 & 0 & 0 & 0 & 0 & 2 & -1 \\ 8| & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{array} \tag{4.10}$$

Properties of $L(G)$

- $L(G)$ symmetric
- the unit vector $e = [1, \dots 1] \in \mathcal{N}(L(G))$ is in the nullspace of the Laplacian matrix
- $\forall \lambda \in \sigma(L(G) > 0$, i.e. all the eigenvalues of $L(G)$ are positive except λ_0 corresponding to the unit vector

- $\lambda_2 \neq 0$ if the graph is connected, $\lambda_2(L(G))$ is also called the algebraic connectivity of the graph

The main ides of the spectral bisection algorithm is

```
compute \(v_2\) eigenvector corresponding to \(\lambda_2(L(G))\)
for i=1,N
  if v_2(i) < 0 put vertex \(V_i\) in N_{-}
  else put vertex \(V_i\) in N_{-}
```

The eigenvectors and eigenvalues are computed using Lanczos's algorithm. These steps are repeated recursively on each of the two branches of the graph N_-, N_+. This is possible since according to Fiedler's theorems the graph N_- is connected, N_+ connected only if no $v_2(i) = 0$, and for each subgraph G_1 the algebraic connectivities satisfy $\lambda_2(L(G_1)) \leq \lambda_2(L(G))$.

To run the genmap code be sure that the Nek tools are up-to-date and compiled. At command line type: genmap NOTE-If the executables for the tools were not placed in the bin directory(default), include the path to the genmap executable. We give here the output for the .rea file in the Kovasznay example

```
Input (.rea) file name:
kov
Input mesh tolerance (default 0.2):
NOTE: smaller is better, but generous is more forgiving for bad meshes.
0.05
 reading .rea file data ...
 start locglob_lexico:           8         960        7680  0.10000000000000001
 locglob:           1           1        7680
 .....
 locglob:           3        1254        7680
 done locglob_lexico:        1254        1254        7680           8
 start periodic vtx:         960        1254
 done periodic vtx
 start rec_bisect:         960
 done:    0.1%
 .....
 done:   99.4%

 done rec_bisect
writing kov.map
```

The user is prompted for .rea file name and should enter only the prefix of the .rea file. The user is prompted for mesh tolerance value. Typically a value of .05 is sufficient. Increasing or decreasing this value should make very little difference in the mesh generation. However, if given an error from genmap, the tolerance may need to be made slightly more generous.

A successful genmap run will produce a .map file with the proper processor decomposition.

NOTE: For large element counts, it is not uncommon for genmap to be produce a few disconnected sets. These sets are typically under 7 elements large and will not affect optimization of the NEK5000 run. If a disconnected set is produced, genmap will output the following warning to stdout.

```
not connected   NO   NEL  Nsets   Nlarge sets
```

Here, `NO` is the number of elements disconnected from the set of `NEL` elements, `Nsets` is the counter of disconnected sets found, and `Nlarge sets` is the number of sets greater than 64 elements in size. `Nlarge sets` should always be 0. If not, please contact someone on the developer team so we can be sure to have a more optimal partition of your mesh.

Genmap outputs an ordered set of numbers which are organized as follows Line number 1 contains the header `nel, nactive, depth, d2, npts, nrank, noutflow`

- `nel` number of elements
- `nactive` nrank-noutflow
- `depth` floor(log2(nel))
- `d2` 2^{depth}
- `npts` number of corner points (`nel*4` in 2D, `nel*8` in 3D)
- `nrank` number of unique corner points
- `noutflow` number of outflows (not used anymore, is zero)

For the Kovasnay flow on an 8 element mesh with periodic boundary conditions we have `8 12 3 8 32 12 0`

Next we have the data (one line per element, listed in order of global element number) ==== `6 12 11 6 5`

This means that elemnt one (since we are on the first line) belongs to group 6, and this element is given by vertices in unique ordering. The vertices are ordered in symmetric ordering (starting at 1)

3 - 4 | | 1 - 2

To distribute amongst processors, one just takes as many consecutive processors as one wants.

Chapter 5

Performing large scale simulations in Nek5000

5.1 Large scale simulations

The largest simulations performed so far with Nek5000 are in the range of 5×10^6. Performance aspects to keep in mind

- design your SEM-mesh for a polynomial order N=7 or N=9 (lx1 in SIZE)
- turn on dealiasing only if needed and try to minimize the polynomial order used for the fine grid (lxd in SIZE)
- ensure that you have at least 50 elements per MPI process (the more the better)
- explore the pressure solver performance using AMG solver (sweet spot depends on number of processors and elements)
- make sure usrchk() does not contain time consuming operations getting called in the time loop
- enable tuned MxM implementation for your platform (see makenek options)
- turn on residual projection scheme (see .rea file parameters)
- tune your pressure/velocity tolerances (e.g. use 5e-5 for pressure and 1e-8 for velocity solver) having in mind your overall accuracy
- try to maximize the timestep, if needed turn on OIFS scheme with a target Courant number of around 2.0
- use the parallel I/O (MPI-IO or custom kernel) to write checkpoints to disk
- understand where you spend most of the time - turn on solver and MPI timings to monitor solver performance (see makenek options)

In such context it is recommendable to save the .rea file as a binary re2.

Also for input/output it may be necessary to use MPI I/O. In this case the code has to be compiled with MPI I/O, i.e. the line PPLIST="MPIIO" in the makefile should not be commented. For output we may use iofiles, the parameter 65 in the .rea file specifies the number of directories and separate files that have to be created as specified by the user.

For large scale simulations the AMG solver is a better, faster choice for solving the Poisson problem (the default solver is XXT).

AMG solver The code should be compiled once with the settings `AMG=true`, `AMG_ DUMP=true`. In the tools folder of Nek5000 we can find the AMG solver, a Matlab version for the moment which is subject to further integration in the main code. The user should run the script run which will read the AMG dump files and create new ones. The new files are now to be used in the code and with `AMG_ DUMP` commented out the user should recompile and run his Nek5000 version.

The AMG solver is a 3 stage process.

The first step will generate the files needed for the matlab code. Next matlab must run the setup code and generate a set of .dat files. Then nek can run with the .dat files and use the AMG pressure solver.

AMG dump stage

Make sure `IFAMG` and `IFAMG_ DUMP` in makenek are uncommented and set to true Run makenek clean, then makenek <casename> Run Nek (this will produce a set of *.dat files)

MATLAB AMG stage

Move the `amgdmp_ *.dat` files to `nek5_ svn/trunk/tools/amg_ matlab`:

```
mv amgdmp*.dat ../../trunk/tools/amg_ matlab
cd ../../trunk/tools/amg_ matlab
```

Run the script: `tools/amg_ matlab/run` (this may take several hours and will produce set of files)

AMG run stage

Move all *.dat files produced back to your case directory:

```
mv *.dat /path/to/case/dir
```

Comment IFAMG_ DUMP in makenek (IFAMG should still be set to TRUE) Run makenek clean, then run `makenek <casename>` Run Nek (the generated AMG files will be read during the AMG setup phase)

Notes on improving AMG results:

To help speed up the matlab process, try running the 1st stage, the AMG dump stage, with `lx1=3` in the SIZE file. Using a lower lx1 number will create a sparser matrix and thus a speedier matlab resolution. lx1 can be increased when ready to run the 2nd stage, the AMG run stage, after the .dat files are produced.

To increase accuracy in the AMG results, try tightening the tolerances in the run script, in `trunk/tools/amg_ matlab`. Specifically, the first tolerance (default set to 0.5). Lowering this (say, to 0.1), will increase the time the matlab code stage takes, but the result will be a faster convergence in the pressure solves of the AMG run stage.

Size related issues Large simulations require a high number of nodes and thus a high number of variables. So far Nek5000 supports by default 4 byte integers, however the node numbering for big meshes may exceed this size and 8 byte integers may be needed. How is this done?

Table 5.1: Compiler options

name	values	default	description
PPLIST	string		list of pre-processor symbols (BG,MOAB,BLAS_ MXM, MPIIO)
IFMPI	true,false	true	use MPI (needed for a multiprocessor computation)
IFAMG_ DUMP	true,false	false	dump AMG pre-processing files
IFAMG	true,false	false	use AMG as coarse grid solver for pressure preconditioner else XXT
F77	string	mandatory	Fortran compiler (e.g. MPI: mpif77)
CC	string	mandatory	C compiler (e.g. MPI: mpicc)
G	string	optional	optional compilation flags
OPT_ FLAGS_ STD7	string	optional	optimization flags for L1,L2,L3
OPT_ FLAGS_ MAG	string	optional	optimization flags for L4 (highest opt level)
SOURCE_ ROOT	string	mandatory	path of nek5000 source
USR	string	optional	object list of additional files to compile (make intructions (makefile_ usr.inc required)
USR_ LFLAGS	string	optional	optional linking flags
MOAB_ DIR	string	NEK with MOAB	Path to MOAB directories
IFVISIT	true,false	false	Toggles Visit in situ. See Visit_ in_ situ for details
VISIT_ INSTALL	string	VISIT in situ	Path to VISIT install path. See Visit_ in_ situ for details.
VISIT_ STOP	true,false	false	When running VISIT in situ, simulation stops after step 1 to connect VISIT.

If more than 9 passive scalars are needed

Exiting Nek5000 while a batch job in being executed should be done not using `"qdel"` but `echo -1 > ioinfo`.

MAKENEK The shell script makenek is designed to assist the compilation process of NEK5000. The script will create a makefile based on the user settings section in makenek. The GNU gmake utility is used to build NEK5000. Available configurations options:

Binary geometry Reatore2 Jump to: navigation, search

The NEK5000 tool, reatore2 allows users to split an ASCII .rea file to an ASCII .rea and a binary .re2 file. The .re2 file contains the mesh and boundary condition data that is normally written in ASCII in the .rea file. For large simulations, this information can be substantial, so storing it in

binary lowers the memory footprint for the simulation. Running reatore2

Be sure that your nekton tools are up-to-date and compiled. At the command prompt type: reatore2

NOTE-If the executables for the tools were not placed in the bin directory(default), include the path to the reatore2 executable

User is prompted for name of .rea file

-Enter the name to the .rea file, excluding the .rea extenstion

User is prompted for the new files name

-Enter the name for your new files

5.2 Parallelism in Nek5000

The parallelism of Nek5000 is accomplished via domain decomposition methods and a suitable gather-scatter code. All this is implemented in such way that the user does not have to be concerned with the parallelism and only focus on the actual solvers while keepin in mind a few simple rules and routines that switch from local to global and back.

- Locally, the SEM is structured.
- Globally, the SEM is unstructured.
- Vectorization and serial performance derive from the structured aspects of the computation.
- Parallelism and geometric flexibility derive from the unstructured, element-by-element, operator evaluation.
- Elements, or groups of elements are distributed across processors, but an element is never subdivided.

For the most part, the global element numbering is not relevant since Nek5000 assigns it randomly but following certain rules.

There are two types of array sizes, starting with lx1, lelv, etc. which give an upper bound of the arrays. And nx1, nelv, etc. which give the actual number of elements/grid points per processors. For the example in Fig. 5.1 we have

- on proc 0, `nelt=2` (nelt = no elements in temperature domain)
- on proc 1, `nelt=3` (nelv = no elements in fluid domain, usually = nelt)

Arrays `lglel` that distinguish which processor has which elements,

- on proc 0, `nelt=2, lglel=(2,5)`, local element `1->2` and `2->5`
- on proc 1, `nelt=3, lglel=(1,3,4)`, local element `1->1`, `2->3` and `4->3`

Now for global to local we have two common arrays (scaling as `nelgt`, but only two such arrays)

- `gllel=(1,1,2,3,2)`, assigns a global element to its local correspondent, i.e. global element `1->1`, `2->1` and `3->2` etc.

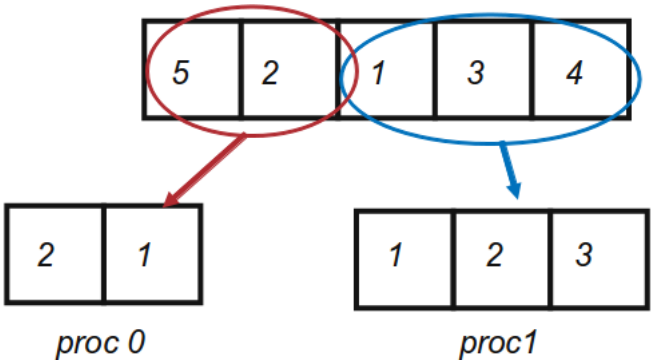


Figure 5.1: A simple SEM row of elements and a potential splitting

- `gllnid=(1,0,1,1,0)`, assigns a global element to its processor, i.e. `1->1`, `2->0` and `3->1` etc.

All data contiguously packed (and quad-aligned) `real u(lx1,ly1,lz1,lelt)` indicates that `u` is a collection of elements, `e=1,...,Nelt =< lelt`, each of size $(N+1)d, d = 2or3$.

Example: Summation

Serial version `s = 0`

```
do e=1,nelv
do iz=1,nz1
do iy=1,ny1
do ix=1,nx1
s=s+u(ix,iy,iz,e)
enddo,...,enddo
```

Second approach, serial version (works in parallel in Nek) `n=nx1*ny1*nz1*nelv`

```
s=0
do i=1,n
s=s+u(i,1,1,1)
enddo
```

Nek Parallel Version

`s=glsum(s,n)`

If you want a local max `s=vlmax(u,n)`, or a global max `s=glmax(u,n)`.

Chapter 6

Routines of interest

The most common routines needed in nek500 are

6.1 Naming conventions

- `subroutine f(a,b,c)` a- returned variable b,c -input data
- `op[]` represent operations on operators
- `c[]` operations on constants
- `gl[]` global operations
- `col2[]` —
- `col3[]` –

6.2 Subroutines

`subroutine rescale_x(x,x0,x1)`

Rescales the array x to be in the range (x0,x1). This is usually called from usrdat2 in the .usr file

`subroutine normvc(h1,semi,l2,linf,x1,x2,x3)`

Computes the error norms of a vector field variable(x1,x2,x3) defined on mesh 1, the velocity mesh. The error norms are normalized with respect to the volume, with the exception on the infinity norm, linf.

`subroutine comp_vort3(vort,work1,work2,u,v,w)`

Computes the vorticity (vort) of the velocity field, (u,v,w)

`subroutine lambda2(l2)`

Generates the Lambda-2 vortex criterion proposed by Jeong and Hussain (1995)

`subroutine planar_average_z(ua,u,w1,w2)`

Computes the r-s planar average of the quantity u.

`subroutine torque_calc(scale,x0,ifdout,iftout)`

Computes torque about the point x0. Here scale is a user supplied multiplier so that the results may be scaled to any convenient non-dimensionalization. Both the drag and the torque can be printed to the screen by switching the appropriate ifdout(drag) or iftout(torque) logical.

`subroutine set_obj`

Defines objects for surface integrals by changing the value of hcode for future calculations. Typically called once within userchk (for istep = 0) and used for calculating torque. (see above)

`subroutine avg1(avg,f, alpha,beta,n,name,ifverbose)`

`subroutine avg2(avg,f, alpha,beta,n,name,ifverbose)`

`subroutine avg3(avg,f,g, alpha,beta,n,name,ifverbose)`

These three subroutines calculate the (weighted) average of f. Depending on the value of the logical, ifverbose, the results will be printed to standard output along with name. In avg2, the f component is squared. In avg3, vector g also contributes to the average calculation.

`subroutine outpost(x,vy,vz,pr,tz,' ')`

Dumps the current data of x,vy,vz,pr,tz to an `.fld` or `.f0????` file for post processing.

`subroutine platform_timer(ivrb)`

Runs the battery of timing tests for matrix-matrix products,contention-free processor-to-processor ping-pong tests, and `mpi_all_reduce` times. Allows one to check the performance of the communication routines used on specific platforms.

`subroutine quickmv`

Moves the mesh to allow user affine motion.

`subroutine runtimeavg(ay,y,j,istep1,ipostep,s5)`

Computes,stores, and (for ipostep!0) prints runtime averages of j-quantity y (along w/ y itself unless ipostep<0) with j + 'rtavg_' + (unique) s5 every ipostep for istep>=istep1. s5 is a string to append to rtavg_ for storage file naming.

`subroutine lagrng(uo,y,yvec,uvec,work,n,m)`

Compute Lagrangian interpolant for uo

`subroutine opcopy(a1,a2,a3,b1,b2,b3)`

Copies b1 to a1, b2 to a2, and b3 to a3, when ndim = 3,

`subroutine cadd(a,const,n)`

Adds const to vector a of size n.

`subroutine col2(a,b,n)`

For n entries, calculates a=a*b.

`subroutine col3(a,b,c,n)`

For n entries, calculates a=b*c.

6.3 Functions

`function glmax(a,n)` `function glamax(a,n)` `function iglmax(a,n)` Calculates the (absolute) max of a vector that is size n. Prefix i implies integer type. `function i8glmax(a,n)` Calculates the max of an integer*8 vector that is size n. `function glmin(a,n)` `function glamin(a,n)` `function iglmin(a,n)` Calculates the (absolute) min of a vector that is size n. Prefix i implies integer type.

`function glsc2(a,b,n)` `function glsc3(a,b,mult,n)` `function glsc23(z,y,z,n)` Performs the inner product in double precision. glsc3 uses a multiplier, mult and glsc23 performs x*x*y*z.

`function glsum(x,n)` `function iglsum(x,n)` `function i8glsum(x,n)` Computes the global sum of x, where the prefix, i specifies type integer, and i8 specifies type integer*8.

6.4 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
```

This example will load a list of field files (filenames are read from a file) into the solver using the `load_fld()` function. After the data is loaded, the user is free to compute other postprocessing quantities. At the end the results are dumped onto a regular (uniform) mesh by a subsequent call to prepost().

Note: The regular grid data (field files) cannot be used as a restart file (uniform->GLL interpolation is unstable)!

```
SUBROUTINE USERCHK
INCLUDE 'SIZE'
INCLUDE 'TOTAL'
INCLUDE 'RESTART'

character*80 filename(9999)

ntot   = nx1*ny1*nz1*nelv

ifreguo = .true.   ! dump on regular (uniform) grid instead of GLL
nrg     = 16       ! dimension of regular grid (nrg**ndim)

! read file-list
if (nid.eq.0) then
   open(unit=199,file='file.list',form='formatted',status='old')
   read(199,*) nfiles
   read(199,'(A80)') (filename(i),i=1,nfiles)
   close(199)
endif
call bcast(nfiles,isize)
call bcast(filename,nfiles*80)

do i = 1,nfiles
   call load\_ fld(filename(i))

   ! do something
   ! note: make sure you save the result into arrays which are
   !       dumped by prepost() e.g. T(nx1,ny1,nz1,nelt,ldimt)
   ...

   ! dump results into file
   call prepost(.true.,'his')
enddo

! we're done
call exitt
```

6.5 Spectral Interpolation Tool

`Check intpts()`. Monitor Points

Multiple monitor points can be defined in the file hpts.in to examine the field data at every timestep.

- setup an ASCII file called 'hpts.in' e.g:

```
3 !number of monitoring points
1.1 -1.2 1.0
. . .
x y z
```

- depending on the number number of monitoring points you may need to increase `lhis` in SIZE.
- add `'call hpts()'` to `userchk()`

6.6 Grid to Grid Interpolation

To interpolate an existing field file (e.g. base.fld) onto a new mesh do the following:

- set lpart in SIZE to a large value (e.g. 100'000 or larger) depending on your memory footprint
- compile Nek with MPIIO support
- set NSTEPS=0 in the .rea file (post-processing mode)
- run nek using the new geometry (e.g. new_geom.f0000)
- run nek using the old geometry and add this code snipplet to userchk()

```
character*132  newfld, oldfld, newgfld
data newfld, oldfld, newgfld /'new0.f0001','base.fld','new\_geom.f0000'/
call g2gi(newfld, oldfld, newgfld) ! grid2grid interpolation
call exitt()
```

6.7 Lagrangian Particle Tracking

The interpolation tool can be used for Lagrangian particle tracking (the particles are the interpolation points).

Workflow: Set initial particle positions (e.g. reading a file particle.pos0) x_part <- x_pos0
LOOP

- compute field quantities
- interpolate field quantities for all particles using intpts()
- dump/store particle data
- advect particles using particle_advect()

END LOOP

```
      subroutine particle\_ advect(rtl,mr,npart,dt\_ p)
c
c     Advance particle position in time using 4th-order Adams-Bashford.
c     U[x\_ i(t)] for a given x\_ i(t) will be evaluated by spectral interpolation.
c     Note: The particle timestep dt\_ p has be constant!
c
      include 'SIZE'
      include 'TOTAL'

      real rtl(mr,1)

      real vell(ldim,3,lpart)  ! lagged velocities
      save vell

      integer icalld
      save    icalld
      data    icalld /0/

      if(npart.gt.lpart) then
        write(6,*) 'ABORT: npart>lpart - increase lpart in SIZE. ',nid
        call exitt
      endif

      ! compute AB coefficients (for constant timestep)
      if (icalld.eq.0) then
         call rzero(vell,3*ldim*npart) ! k = 1
         c0 = 1.
         c1 = 0.
         c2 = 0.
         c3 = 0.
         icalld = 1
      elseif (icalld.eq.1) then        ! k = 2
         c0 = 1.5
         c1 = -.5
         c2 = 0.
         c3 = 0.
         icalld = 2
      elseif (icalld.eq.2) then        ! k = 3
         c0 = 23.
         c1 = -16.
         c2 = 5.
         c0 = c0/12.
         c1 = c1/12.
         c2 = c2/12.
         c3 = 0.
```

```
   icalld = 3
 else                              ! k = 4
   c0 = 55.
   c1 = -59.
   c2 = 37.
   c3 = -9.
   c0 = c0/24.
   c1 = c1/24.
   c2 = c2/24.
   c3 = c3/24.
 endif

 ! compute new position x[t(n+1)]
 do i=1,npart
   do k=1,ndim
      vv = rtl(1+2*ndim+k,i)
      rtl(1+k,i) =  rtl(1+k,i) +
\&                   dt\_p*(
\&                   + c0*vv
\&                   + c1*vell(k,1,i)
\&                   + c2*vell(k,2,i)
\&                   + c3*vell(k,3,i)
\&                   )
      ! store velocity history
      vell(k,3,i) = vell(k,2,i)
      vell(k,2,i) = vell(k,1,i)
      vell(k,1,i) = vv
   enddo
 enddo

 return
 end
```

Chapter 7

Appendix

7.1 Appendix A. Extensive list of parameters .rea file

7.1.1 Parameters

This section tells nek5000

- If the input file reflects a 2D or 3D job (it should match the *ldim* parameter in the SIZE file).
- The combination of heat transfer, Stokes, Navier-Stokes, steady or unsteady to be run.
- The relevant physical parameters.
- The solution algorithm within nek5000 to use.
- The timestep size or Courant number to use, or whether to run variable DT (dt), etc.

A .rea file starts with the following three parameters:

NEKTON VERSION the version of nek5000

DIMENSIONAL RUN number of spatial dimensions (NDIM=2,3 - has to match the setting in the SIZE file).

PARAMETERS FOLLOW the number of parameters which are going to be followed in the .rea file.(NPARAM)

The latter specifies how many lines of .rea file, starting from the next line, are the parameters and have to be read by the program.

7.1.2 Available Parameters

P001 DENSITY density for the case of constant properties (for variable density see parameter **P030**).

P002 VISCOS kinematic viscosity (if $< 0 \rightarrow Re$, otherwise $1/Re$).

P003 BETAG if > 0, natural convection is turned on (Boussinesq approximation). NOT IN USE !

P004 GTHETA model parameter for Boussinesq approximation (see parameter P003). NOT IN USE!

P005 PGRADX NOT IN USE!

P006 NOT IN USE!

P007 RHOCP navier5.f: param(7) = param(1) ! rhoCP = rho NOT IN USE!

P008 CONDUCT conductivity for the case of constant properties (if < 0, it defines the Peclet number, see parameter P030).
connect2.f: if(param(8) .lt.0.0) param(8) = -1.0/param(8)
navier5.f: param(8) = param(2) ! conduct = dyn. visc

P009 NOT IN USE! (passed to CPFLD(2,3)!)
connect2.f: CPFLD(2,3)=PARAM(9)

P010 FINTIME if > 0, specifies simulation end time. Otherwise, use NSTEP (P011).
drive2.f: FINTIM = PARAM(10)

P011 NSTEP number of time steps.
connect2.f: param(11) = 1.0
drive2.f: NSTEPS = PARAM(11)

P012 DT upper bound on time step dt (if < 0, then $dt = |P012|$ constant)
connect2.f: param(12) = 1.0
drive2.f: DT = abs(PARAM(12))

P013 IOCOMM frequency of iteration histories
drive2.f: IOCOMM = PARAM(13)

P014 IOTIME if > 0, time interval to dump the fld file. Otherwise, use IOSTEP (P015).
drive2.f: TIMEIO = PARAM(14)

P015 IOSTEP dump frequency, number of time steps between dumps.
drive2.f: IOSTEP = PARAM(15)
navier5.f: if (iastep.eq.0) iastep=param(15) ! same as iostep

P016 PSSOLVER heat/passive scalar solver:

1: Helmholz

2: CVODE

3: CVODE with user-supplied Jacobian

Note: a negative number will set source terms to 0.

P017 AXIS NOT IN USE!

P018 GRID NOT IN USE!

P019 INTYPE NOT IN USE!
connect2.f: param(19) = 0.0

P020 NORDER NOT IN USE!

P021 DIVERGENCE tolerance for the pressure solver.
drive2.f: TOLPDF = abs(PARAM(21))
hmholtz.f: if (name.eq.'PRES'.and.param(21).ne.0) tol=abs(param(21))

P022 HELMHOLTZ tolerance for the velocity solver.
drive2.f: TOLHDF = abs(PARAM(22))
hmholtz.f: if (param(22).ne.0) tol=abs(param(22))
hmholtz.f: if (param(22).lt.0) tol=abs(param(22))*rbn0
navier4.f: if (param(22).ne.0) tol = abs(param(22))

P023 NPSCAL number of passive scalars.
connect2.f: NPSCAL=INT(PARAM(23))

P024 TOLREL relative tolerance for the passive scalar solver (CVODE).
drive2.f: TOLREL = abs(PARAM(24))

P025 TOLABS absolute tolerance for the passive scalar solver (CVODE).
drive2.f: TOLABS = abs(PARAM(25))

P026 COURANT maximum Courant number (number of RK4 substeps if OIFS is used).
drive2.f: CTARG = PARAM(26)

P027 TORDER temporal discretization order (2 or 3).
drive2.f: NBDINP = PARAM(27)

P028 NABMSH Order of temporal integration for mesh velocity.if 1, 2, or 3 use Adams-Bashforth of corresponding order. Otherwise, extrapolation of order TORDER (P027).

P029 MHD_VISCOS if $> 0 \rightarrow$ magnetic viscosity, if $< 0 \rightarrow$ magnetic Reynolds number.
connect2.f: if(param(29).lt.0.0) param(29) = -1.0/param(29)
connect2.f: if (param(29).ne.0.) ifmhd = .true.
connect2.f: cpfld(ifldmhd,1) = param(29) ! magnetic viscosity

P030 USERVP if

0: constant properties

1: user-defined properties via USERVP subroutine (each scalar separately)

2: user-defined properties via USERVP subroutine (all scalars at once)

P031 NPERT if $\neq 0$, number of perturbation modes in linearized N-S.
connect2.f: if (param(31).ne.0.) ifpert = .true.
connect2.f: if (param(31).lt.0.) ifbase = .false. ! don't time adv base flow
connect2.f: npert = abs(param(31))

P032 NBCRE2 if > 0, number of BCs in .re2 file, 0: all.
connect2.f: if (param(32).gt.0) nfldt = ibc + param(32)-1

P033 NOT IN USE!

P034 NOT IN USE!

P035 NOT IN USE!

P036 XMAGNET NOT IN USE!

P037 NGRIDS NOT IN USE!

P038 NORDER2 NOT IN USE!

P039 NORDER3 NOT IN USE!

P040 NOT IN USE!

P041 $1 \rightarrow$ multiplicative SEMG
hsmg.f:c if (param(41).eq.1) ifhybrid = .true. $\leftarrow$ NOT IN USE!

P042 linear solver for the pressure equation, $0 \rightarrow$ GMRES or $1 \rightarrow$ PCG

P043 0: additive multilevel scheme - 1: original two level scheme.
navier6.f: if (lx1.eq.2) param(43)=1.
navier6.f: if (param(43).eq.0) call hsmg_setup

P044 0=E-based additive Schwarz for PnPn-2; 1=A-based.

P045 Free-surface stability control (defaults to 1.0)
subs1.f: FACTOR = PARAM(45)

P046 if > 0, do not set Initial Condition (no call to subroutine SETICS).
drive2.f: irst = param(46)
ic.f: irst = param(46) ! for lee's restart (rarely used)
subs1.f: irst = param(46)

P047 parameter for moving mesh (Poisson ratio for mesh elasticity solve (default 0.4)).
mvmesh.f: VNU = param(47)

P048 NOT IN USE!

P049 if < 0, mixing length factor NOT IN USE!.
drive2.f:c IF (PARAM(49) .LE. 0.0) PARAM(49) = TLFAC
turb.f: TLFAC = PARAM(49)

P050 NOT IN USE!

P051 NOT IN USE!

P052 HISTEP if > 1, history points dump frequency (in number of steps).
prepost.f: if (param(52).ge.1) iohis=param(52)

P053 NOT IN USE!

P054 direction of fixed mass flowrate (1: x-, 2: y-, 3: z-direction). If 0: x-direction.
drive2.f: if (param(54).ne.0) icvflow = abs(param(54))
drive2.f: if (param(54).lt.0) iavflow = 1 ! mean velocity

P055 volumetric flow rate for periodic case; if p54< 0, then p55:=mean velocity.
drive2.f: flowrate = param(55)

P056 NOT IN USE!

P057 NOT IN USE!

P058 NOT IN USE!

P059 if $\neq 0$, deformed elements (only relevant for FDM). !=0 $\rightarrow$ full Jac. eval. for each el.

P060 if $\neq 0$, initialize velocity to 1e-10 (for steady Stokes problem).

P061 NOT IN USE!

P062 if > 0, swap bytes for output.

P063 WDSIZO real output wordsize (8: 8-byte reals, else 4-byte).
prepost.f: if (param(63).gt.0) wdsizo = 8 ! 64-bit .fld file

P064 if $= 1$, restart perturbation solution
pertsupport.f: if(param(64).ne.1) then !fresh start, param(64) is restart flag

P065 number of IO nodes (if < 0 write in separate subdirectories).

P066 Output format: (only postx uses .rea value; other nondefault should be set in usrdat) (if ≥ 0 binary else ASCII).
connect2.f: param(66) = 6 ! binary is default
connect2.f: param(66) = 0 ! ASCII

P067 read format (if ≥ 0 binary else ASCII).

P068 averaging frequency in avg_all (0: every timestep).

P069 NOT IN USE!

P070 NOT IN USE!

P071 NOT IN USE!

P072 NOT IN USE!

P073 NOT IN USE!

P074 if > 0 print Helmholtz solver iterations.
hmholtz.f: if (nid.eq.0.and.ifprint.and.param(74).ne.0) ifprinthmh=.true.

P075 NOT IN USE!

P076 NOT IN USE!

P077 NOT IN USE!

P078 NOT IN USE!

P079 NOT IN USE!

P080 NOT IN USE!

P081 NOT IN USE!

P082 coarse-grid dimension (2: linear). NOT IN USE!

P083 NOT IN USE!

P084 if < 0, force initial time step to this value.

P085 set *dt* in *setdt*.
subs1.f: dt=dtopf*param(85)

P086 RESERVED ! if $\neq 0$, use skew-symmetric form, else convective form.
drive2.f: PARAM(86) = 0 ! No skew-symm. convection for now
navier1.f: if (param(86).ne.0.0) then ! skew-symmetric form

P087 NOT IN USE!

P088 NOT IN USE!

P089 RESERVED !

P090 NOT IN USE!

P091 NOT IN USE!

P092 NOT IN USE!

P093 number of previous solutions to use for residual projection.
(adjust MXPREF in SIZEu accordingly)

P094 if > 0, start projecting velocity and passive scalars after P094 steps

P095 if > 0, start projecting pressure after P095 steps

P096 NOT IN USE!

P097 NOT IN USE!

P098 NOT IN USE!

P099 dealiasing:

< 0: disable

3: old dealiasing

4: new dealiasing

P100 RESERVED ! pressure preconditioner when using CG solver (0: Jacobi, > 0: two-level Schwarz) or wiseversa?

P101 number of additional modes to filter (0: only last mode)
navier5.f: ncut = param(101)+1

P102 NOT IN USE!

P103 filter weight for last mode (< 0: disabled)

P107 if $\neq 0$, add it to h2 in sethlm

P116 NELX number of elements in x for Fast Tensor Product FTP solver (0: do not use FTP). NOTE: box geometries, constant properties only!

P117 NELY number of elements in y for FTP

P118 NELZ number of elements in z for FTP

7.1.3 Available Logical Switches

This part of .rea file starts with such a line:

```
n   LOGICAL SWITCHES FOLLOW
```

where n is the number of logical switches which is set in the following lines.

7.1.4 Logical switches

Note that by default all logical switches are set to false.

IFFLOW solve for fluid (velocity, pressure).

IFHEAT solve for heat (temperature and/or scalars).

IFTRAN solve transient equations (otherwise, solve the steady Stokes flow).

IFADVC specify the fields with convection.

IFTMSH specify the field(s) defined on T mesh (first field is the ALE mesh).

IFAXIS axisymmetric formulation.

IFSTRS use stress formulation in the incompressible case.

IFLOMACH use low Mach number compressible flow.

IFMGRID moving grid (for free surface flow).

IFMVBD moving boundary (for free surface flow).

IFCHAR use characteristics for convection operator.

IFSYNC use mpi barriers to provide better timing information.

IFUSERVP user-defined properties (e.g., μ, ρ) varying with space and time.

7.2 Appendix B. Extensive list of parameters SIZE file

ldim : number of spatial dimensions (2 or 3).

lx1, ly1, lz1 : number of (GLL) points in the x, y and z directions, respectively, within each element of mesh1 (velocity) which is equal to the (polynomial order+1) by definition. $ly1$ is usually the same as $lx1$ and for 2D cases $lz1 = 1$.
[1] [2]

lx2, ly2, lz2 : number of (GLL) points in the x, y and z directions, respectively, within each element of mesh2 (pressure). Use $lx2 = lx1$ for PN/PN formulation or $lx2 = lx1 - 2$ for PN/PN-2 formulation.

lx3, ly3, lz3 : number of (GLL) points in the x, y and z directions, respectively, within each element of mesh3.
[3]

lxd, lyd, lzd : number of points for over integration (dealiasing), use three half rule e.g. for $lx1 = 8$ use $lxd = 12$.

lelx, lely, lelz : maximum number of elements per rank for global FDM (Fast Diagonalization Method) solver.

ldimt : maximum number of T-array fields (temperature + additional scalars).

lp : maximum number of ranks.

lelg : maximum (global) number of elements (it is usually set more than the # of elements existing in the mesh, for making maximum use of memory is can be set to the exact number of mesh elements).

lelt : maximum number of local elements for T-mesh (per rank, $lelt \geq lelg/np + 1$).

[1] is $lx1 \neq ly1$ supported?
[2] $lx1$ recomeneded odd for better performance
[3] mesh3 is rarely used

lelv : maximum number of local elements for V-mesh ($lelv = lelt$).

lpelv,lpelt,lpert : Number of elements of the perturbation field, number of perturbation fields

lpx1, lpy1, lpz1 : Number of point in x,y,z direction of perturbation field within each element of mesh1

lpx2, lpy2, lpz2 : Number of point in x,y,z direction of perturbation field within each element of mesh2

lbelv, lbelt : Total Number of elements of the B-field (MHD)

lbx1, lby1, lbz1 : Number of point in x,y,z direction of B-field within each element of mesh1

lbx2, lby2, lbz2 : Number of point in x,y,z direction of B-field within each element of mesh2

lx1m, ly1m, lz1m : when the mesh is a moving type $lx1m = lx1$, otherwise it is set to 1.

lxz : LXZ = LX1*LZ1

```
connect1.f:  common /scruz/ snx(lxz) , sny(lxz) , snz(lxz) , efc(lxz)
```

lorder : maximum time integration order (2 or 3).

maxobj : maximum number of objects. zero if not using objects?

maxmbr : maximum number of members in an object.

lhis : maximum number of history points a single rank will read in (NP*LHIS < number of points in hpts.in).

lctmp0 : NOT IN USE!

```
drive1.f:c COMMON /CTMP0/ DUMMY0(LCTMP0)
```

lctmp1 :

```
drive1.f:c COMMON /CTMP1/ DUMMY1(LCTMP1)
drive2.f:  COMMON /SCRNS/ WORK(LCTMP1)
```

lvec : NOT IN USE!

mxprev : maximum number of history entries for residual projection (recommended value: 20).

lgmres : dimension of Krylov subspace in GMRES (recommended value: 40).

lmvec : NOT IN USE !

lsvec : NOT IN USE!

lstore : NOT IN USE!

maxmor : $= lelt$

lzl : for 2D cases $lzl = 1$ and for 3D cases $lzl = 3$ (computed automatically).

The following parameters are deprecated and were subsequently removed in newer versions.

LELGEC : LELGEC = 1

LXYZ2 : LXYZ2 = 1

LXZ21 : LXZ21 = 1

LMAXV : LMAXV = LX1*LY1*LZ1*LELV

LMAXT : LMAXT = LX1*LY1*LZ1*LELT

LMAXP : LMAXP = LX1*LY1*LZ1*LELV

Bibliography

[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.