Contents

Chapter 1. Introduction

This manual provides an introduction to the use of Nek5000, which is a simulation code for analyzing unsteady incompressible fluid flow with thermal and passive scalar transport. Nek5000 can treat general two- and three-dimensional domains described by isoparametric quad or hex elements. In addition, it can be used to compute axisymmetric flows. It is a time-stepping based code and does not currently support steady-state solvers, other than steady Stokes flow and steady heat conduction.

Nek5000 is based on the Nekton 2.0 spectral element code written by Paul Fischer, Lee Ho, and Einar Rønquist in 1986-1991, under direction and theoretical guidance from Tony Patera and Yvon Maday. Nekton 2.0 was the first three-dimensional spectral element code and one of the first commercially available codes for distributed-memory parallel processors. Nek5000 development was continued from the 90s through the present by Paul Fischer, Jerry Kruse, Julie Mullen, Henry Tufo, James Lottes, Stefan Kerkemeier, Katie Heisey, and Ananias Tomboulides.

Chapter 2. Quick Start

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

2.1 Download and Initial Build

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. [1]

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.

[1] In some cases it may be necessary to reduce the memory footprint of some of the tools. The procedures are (will be) described in SEC. TROUBLESHOOTING.

2.2 Running the First 2D Example

As a first example, we consider the eddy problem due to Walsh [2] To get started, execute the following commands,

```
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.) 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
```

which, if all works properly, will generate the executable `nek5000` 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
```

[2]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)

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. See Section 7 for a comprehensive discussion of timestep selection.

2.3 Viewing the First 2D Example

The preferred mode for data visualization and analysis with Nek5000 is to use VisIt, which is described in Section ??. 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.

To see the vorticity at the final time, load the last output file, `eddy_uv.fld12`, by clicking/typing the following in the postx window:

	click	**type**	**comment**
1.	SET TIME	12	load fld12
2.	SET QUANTITY		
3.	VORTICITY		
4.	PLOT		

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 through the following sequence:

	click	**type**	**comment**
1.	SET TIME	11	load fld11
2.	SET QUANTITY		
3.	VELOCITY		
4.	MAGNITUDE		
5.	PLOT		

2.4 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

```
cpn 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.fld11
```

which tells nek5000 to use the contents of `eddy_uv.fld11` as the initial condition for `eddy_new`. (**Don't** use fld12 – that has vorticity in it because of the `outpost` call placed in `userchk`.) 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 (see Section **??**). 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.

Chapter 3. Basic Problem Setup

Nek5000 consists of three principal modules: the preprocessor `prex`, the solver `nek5000`, and the postprocessor `postx`. `prex` and `postx` are based upon an X-windows GUI. Nek5000 output formats can also be read by the parallel visualization package VisIt developed by Hank Childs and colleagues at LLNL/LBNL. VisIt is mandatory for large problems (e.g., more than 100,000 spectral elements). 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.

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`. Suppose you are doing a calculation called "shear." The key files defining the problem would be shear.rea, shear.map, and shear.usr. SIZE controls (at compile time) the polynomial degree used in the simulation, as well as the space dimension $d = 2$ or 3.

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

3.1 Contents of .rea file

The `.rea` file consists of several sections:

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

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

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

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

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.

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.

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 essential for i/o performance when the number of elements is large (> 100000).

3.2 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.

3.3 Contents of SIZE file

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*.

3.4 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 $P \geq 19,200\text{MB}/(400\text{MB/proc}) = 48$ 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 admissable number of processors ($P_{\text{mem}} \leq P \leq E$, where P_{mem} is the value computed above).

Chapter 4. Simulation Capabilities

This chapter describes the capabilities of **Nek5000** by introducing the set of governing equations together with boundary conditions, initial conditions, and material property/forcing functions that the program can solve numerically.

4.1 Governing Equations

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), described in Section **??**.

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. The solution variables are the fluid velocity $\mathbf{u} = (u_x, u_y, u_z)$, the pressure p, the temperature T, independent passive scalar fields ϕ_i, i=1,2,..., mesh velocity $\mathbf{w} = (w_x, w_y, w_z)$ (for time-dependent geometry), and magnetic field $\mathbf{B} = (B_x, B_y, B_z)$ (for MHD). 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. 4.1 The governing equations used are: the momentum equations in Ω_f,

$$\rho(\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u}) = -\nabla p + \nabla \cdot [\mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)] + \rho \mathbf{f} \ , \tag{4.1}$$

the continuity (mass conservation) equation in Ω_f,

$$\nabla \cdot \mathbf{u} = 0 \ , \tag{4.2}$$

the energy equation in $\Omega_f \cup \Omega_s$,

$$\rho c_p(\partial_t T + \mathbf{u} \cdot \nabla T) = \nabla \cdot (k \nabla T) + q_{vol} \ , \tag{4.3}$$

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

$$(\rho c_p)_i(\partial_t \phi_i + \mathbf{u} \cdot \nabla \phi_i) = \nabla \cdot (k_i \nabla \phi_i) + (q_{vol})_i. \tag{4.4}$$

Figure 4.1: Computational domain showing respective fluid and solid subdomains, Ω_f and Ω_s.

The above governing equations are subject to boundary conditions and initial conditions described in the following sections.

It should be noted that Nek5000 can solve not only the above fully coupled system, but also various subsets of these equations. Namely, Nek5000 recognizes the following subsets.

1. Fluid flow only; i.e., the momentum and continuity equations only.
2. Fluid flow and heat transfer only; i.e., the momentum, continuity and energy equations only.
3. Fluid flow, heat transfer, and an arbitrary subset of passive scalar fields; i.e., the momentum, continuity, energy and convective-diffusive equations for ϕ_i
4. Fluid flow with the nonstress formulation; i.e., the momentum equation, in which the term $(\nabla\mathbf{u})^T$ in the viscous contribution is set to zero, the continuity equation, and with or without the energy and convective-diffusive equations.
5. Low Mach-number flows where density may vary due to thermal-dilation effects.
6. Unsteady Stokes flow; i.e., the momentum equation, in which the nonlinear term $\mathbf{u}\cdot\nabla\mathbf{u}$ is set to zero, and the continuity equation.
7. Steady Stokes flow; i.e., the momentum and continuity equations with the nonlinear term $\mathbf{u}\cdot\nabla\mathbf{u}$ and transient term $\partial_t\mathbf{u}$ set to zero.
8. Transient heat transfer; i.e., the energy equation only with prescribed steady or unsteady velocity.
9. Steady conduction heat transfer; i.e., only the energy equation in which the advection term $\mathbf{u}\cdot\nabla T$ and the transient term $\partial_t T$ are zero.

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. Using the minimum set of equations possible for a given application clearly saves computational time as Nek5000 solves only those equations specified.

4.1.1 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*

$$\rho(\partial_t \mathbf{u}_i' + \mathbf{u} \cdot \nabla \mathbf{u}_i' + \mathbf{u}_i' \cdot \nabla \mathbf{u}) = -\nabla p_i' + \mu \nabla^2 \mathbf{u}_i', \qquad \nabla \cdot \mathbf{u}_i' = 0, \quad (4.5)$$

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.

4.2 Geometry

4.2.1 Convention and Restrictions

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. 2.1 for several different geometries. 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 $\partial\Omega_f'$, and that part of the boundary of Ω_f which is shared by Ω_s as $\partial\overline{\Omega}_f'$ (note $\partial\Omega_f = \partial\Omega_f' \cup \partial\overline{\Omega}_f'$). In addition, $\partial\Omega_s$, $\partial\Omega_s'$ and $\partial\overline{\Omega}_s'$ are analogously defined. These distinct portions of the domain boundary are illustrated in Fig. 2.1. 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.

- In the geometry generation using PRENEK, a three-dimensional geometry must correspond to a series of layers of planar two-dimensional geometries. Each of those layers must adhere to the rules governing planar two-dimensional geometries. In addition, it is assumed that the parallel planes which seperate the layers of elements are oriented parallel to the x-y plane, and that the first such plane is located at $z = 0$. Subsequent planes are located at arbitrarily increasing values of z. Note that this is a PRENEK restriction as Nek5000 can readily accept general three-dimensional geometries.

Fully general three-dimensional meshes generated by other softwares packages can be input to PRENEK as import meshes. Currently, meshes generated by PRE-BFC and I-DEAS can be accepted by PRENEK.

4.2.2 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 **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 are 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. 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 responsibity 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.

4.3 Boundary Conditions

The boundary conditions for the governing equations given in the previous section are described in the following. Note that if the boundary conditions (for any field variable) are nonzero, the inhomogeneities can either be defined as constant, or as fortran functions of appropriate parameters such as space, time, temperature, etc. In this case, the user is responsible for using relevant variables in all fortran function definitions.

4.3.1 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 traction is specified. For segments that constitute the boundary $\partial\Omega_f$ (refer to Fig. 2.1), 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 Dirchlet, 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. 2.2. 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.

4.3.2 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.

4.3.3 Passive scalar

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. 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.

4.3.4 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. 2.1) 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 tempertaure 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 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 Material Properties and Forcing Functions

The following restrictions are imposed on the material property and forcing function parameters in the governing equations and initial and boundary conditions described in the above sections (here and in what follows time-independent implies no variation in time, whereas constant refers to uniform in space).

- 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 strainrate 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.

- $\mathbf{f}(t)$, 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 $length^2/time$.

Chapter 5. Nek5000 Variables

This chapter identifies data layout within Nek5000 along with key variables that are typically interrogated for analysis.

To begin, we consider the basic notation associated with the theory. In the spectral element method (SEM), data is represented on sets of non-overlapping subdomains, Ω^e, $e = 1, \ldots, E$, with the entire domain $\Omega := \bigcup \Omega^e$. For conjugate heat transfer applications comprising solid and fluid subdomains, we denote

$$\Omega_f := \bigcup_{e=1}^{E_f} \Omega^e, \qquad \Omega \;=\; \Omega_t := \bigcup_{e=1}^{E} \Omega^e, \tag{5.1}$$

where Ω_f denotes the fluid-only domain and $\Omega_t (\equiv \Omega)$ denotes the thermal domain. Note that the fluid domain is always a subset of the thermal domain and, because they are numbered first, fluid elements can be identified with a restriction $e \in [1, E_f]$.

On each element, a typical variable $u(\mathbf{x})$ has the representation

$$u(\mathbf{x})|_{\Omega^e} \;=\; u(x,y,z)|_{\Omega^e} \;=\; \sum_{k=0}^{N}\sum_{j=0}^{N}\sum_{i=0}^{N} u^e_{ijk}\, h_i(r)\, h_j(s)\, h_k(t), \tag{5.2}$$

where u^e_{ijk} are the basis coefficients (typically, the unknown velocity or temperature values); $h_i(r)$, $h_j(s)$, $h_k(t)$ are the Nth-order 1-D Lagrange polynomials based on the GLL[1] points; and $\mathbf{r} := (r,s,t)$ are the coordinates in the canonical reference element, $\hat{\Omega} := [-1,1]^3$. (There should be no confusion with t, the time variable, as the distinction will be clear from context.) Here, we also assume that $\mathbf{x} := (x,y,z) \in \Omega^e$ is given by the isoparametric mapping,

$$\mathbf{x}|_{\Omega^e} \;=\; \mathbf{x}^e \;\;=\;\; \sum_{k=0}^{N}\sum_{j=0}^{N}\sum_{i=0}^{N} \mathbf{x}^e_{ijk}\, h_i(r)\, h_j(s)\, h_k(t). \tag{5.3}$$

That is, Ω^e is the image of $\hat{\Omega}$ under the polynomial mapping (5.3). We further assume that the mapping is invertible, which implies that angles at the corners in Ω^e are bounded away from 0 and 180 degrees. Because of the polynomial map (5.3), it is also important that the edges of Ω^e be smooth. Corners can be accommodated if they coincide with element vertices.

[1] Gauss-Lobatto-Legendre quadrature points.

5.1 Data Layout

With these preliminaries, we turn to the variable and data layout. We note that almost all of the variables in Nek5000 can be accessed via the include statements:

```
include 'SIZE'
include 'TOTAL'
```

within any subroutine. To illustrate the data layout, we consider the x-component of velocity, which has the declaration

```
real vx(lx1,ly1,lz1,lelt)
```

Here, the (fortran) parameters `lx1=ly1=lz1` correspond to $N + 1$ (one greater than the polynomial order), and `lelt` corresponds to an *upper bound* on the number of elements per processor (in fact, per mpi process, but we will use 'processor' throughout to emphasize the notion of distinct, independent, computing resources).

For a *given problem*, the number of elements (E_f, E_t) is specified by `nelv,nelt` in the .rea (or, if present, .re2) file. However, when processing in parallel, `nelv,nelt` represent the number of fluid/thermal elements assigned to the given processor. Thus, we have one definition of `nelv/nelt` associated with the problem, and one associated with the subproblem running on each processor. When running in serial, the two are coincident. Note that, in each case, `nelv` $\leq$ `nelt` $\leq$ `lelt`, and the fluid elements are ordered first on each processor.

For performance reasons, data in Nek5000 is contiguous in memory, with $u^e_{0,0,0}$ followed by $u^e_{1,0,0}$, and so on, up to with $u^e_{N,N,N}$ [:=`u(lx1,ly1,lz1,e)`]. Because fortran passes data by reference, one can pass the contents of the eth element of u to a routine with a call of the following form

```
call my_routine(u(1,1,1,e),...)
```

In addition, one can access the contents of, say, `vx` via the following loop structure:

```
include 'SIZE'
include 'TOTAL'
:
:
```

```
n = nx1*ny1*nz1*nelv
sum=0
do i=1,n
   sum = sum+vx(i,1,1,1)
enddo
```

which would provide the local sum of `vx` on the given processor. Note that the sum, as written, would be different on each processor. To get the full sum across all processors, one would add

```
sum = glsum(sum,1)  ! global sum
```

after then `enddo` statement in the preceding code segment. In this case, the entire operation could also be written as

```
n = nx1*ny1*nz1*nelv
sum = glsum(vx,n)  ! global sum
```

which computes the vector reduction within each processor followed by the requisite global sum across all processors. The result is returned to each processor. Note that `glsum` requires participation of each processor and thus must not be put in a loop whose length varies from processor to processor (e.g., not in a loop of length `nelv`). On most parallel machines (with the exception of the Blue Gene series), the cost of vector reductions scales as $\alpha \log_2 P$, where P is the number of processors and α is the communication latency. It is thus relatively expenseive.

It is worth noting that a proper average of `vx`, defined as

$$\bar{u} := \frac{\int_\Omega u \, dV}{\int_\Omega 1 \, dV} \tag{5.4}$$

would be computed as

```
n = nx1*ny1*nz1*nelv
ubar = glsc2(bm1,vx,n)/volvm1
```

Here, the `'m1'` suffix refers to Mesh 1, which is where velocity and temperature are represented. Because Nek5000 supports a discontinous pressure of order $N-2$ (for the $\mathbb{P}_N - \mathbb{P}_{N-2}$ formulation), there is also a set of variables represented on Mesh 2. Specifically, the pressure has a declaration of the form

```
real pr(lx2,ly2,lz2,lelv)
```

with `lx2=lx1-2` declared in `SIZE`. If one is using the continuous-pressure $\mathbb{P}_N - \mathbb{P}_N$ formulation, then `lx2=lx1`.

There are several variables that have dual representation, including the geometry,

```
xm1(lx1,ly1,lz1,lelv)  ! x on mesh 1
ym1(lx1,ly1,lz1,lelv)  ! etc.
zm1(lx1,ly1,lz1,lelv)

xm2(lx2,ly2,lz2,lelv)  ! x on mesh 2
ym2(lx2,ly2,lz2,lelv)  ! etc.
zm2(lx2,ly2,lz2,lelv)
```

the mass matrices,

```
bm1(lx1,ly1,lz1,lelv)  ! B on mesh 1
bm2(lx2,ly2,lz2,lelv)  ! B on mesh 2
```

and to domain volumes, `volvm1, voltm1, volvm2, voltm2`, which are the respective volumes for Ω_f and Ω_t represented on Mesh 1 and Mesh 2.

5.2 Variables

This section provides a listing of several of the key variables in Nek5000. We begin with the principal dependent variables followed by independent variables, although it worth remarking that geometry is a dependent variable in the case of free-surface and other moving-mesh simulations.

Velocity:	`vx(lx1,ly1,lz1,lelv), vy(lx1,ly1,lz1,lelv), vz(lx1,ly1,lz1,lelv)`
Pressure:	`pr(lx2,ly2,lz2,lelv), pm1(lx1,ly1,lz1,lelv)`
Temperature and Passive Scalars:	`t(lx1,ly1,lz1,lelt,ldimt)`
Mass Matrices:	`bm1(lx1,ly1,lz1,lelt), bm2(lx2,ly2,lz2,lelt)`
Geometry:	`xm1(lx1,ly1,lz1,lelt), ym1(lx1,ly1,lz1,lelt), zm1(lx1,ly1,lz1,lelt)`

`xm2(lx1,ly1,lz1,lelt), ym2(lx1,ly1,lz1,lelt), zm2(lx1,ly1,lz1,lelt)`

In addition to volumetric arrays of the types listed above, there are several surface arrays associated with geometry that are convenient for computing quantities involving surface integrals such as drag and thermal fluxes. Four of the most important arrays are the surface Jacobian (`area()`) and the associated components of the unit normal vectors, which point outward away from the given element. The following lists the data layout for these arrays.

Surface Geometry: `area(lx1,lz1,6,lelt)`
`unx(lx1,lz1,6,lelt), uny(lx1,lz1,6,lelt), unz(lx1,lz1,6,lelt)`
The ordering of the surface arrays follows the lexicographical ordering of the volumetric data. A typical access pattern for surface data is shown in the following example, which calculates the net and average flux through the aggregate set of faces associated with the "t" boundary condition.

```
      subroutine my_flux_calc
      common /mystuff/ tx(lx1,ly1,lz1,lelt)
     $                , ty(lx1,ly1,lz1,lelt)
     $                , tz(lx1,ly1,lz1,lelt)

      integer e,f
      nface = 2*ndim
      a = 0.
      s = 0.
      call gradm1(tx,ty,tz,t) ! grad T
      do e=1,nelv
      do f=1,nface
         if (cbc(f,e,2).eq.'t  ') then
            call facind(i0,i1,j0,j1,k0,k1,nx1,ny1,nz1,f)
            l=0
            do k=k0,k1 ! March over face f
            do j=j0,j1
            do i=i0,i1
               l = l + 1
               s = s + (unx(l,1,f,e)*tx(i,j,k,e)
     $                +  uny(l,1,f,e)*ty(i,j,k,e)
     $                +  unz(l,1,f,e)*tz(i,j,k,e))*area(l,1,f,e)
               a = a + area(l,1,f,e)
            enddo
            enddo
            enddo
         endif
```

```
  enddo
  enddo
  a=glsum(a,1)     ! Sum across processors
  s=glsum(s,1)
  abar = s/a
  if (nid.eq.0) write(6,1) istep,time,s,a,abar
1 format(i9,1p4e13.5,' net flux')
  return
  end
```

Chapter 6. Nek5000 Usage

6.1 Setting Initial Conditions

If not using *restart* (Sec. ??), initial conditions are defined in the `.usr` file within the `useric()` routine, which is called once for each gridpoint, for each field (`ifield`=1,2,. . . , for velocity, temperature, passive scalar 1, etc.). It is important to note that the number of calls to `useric` is large and can vary from one processor to the next because of variability in the number of elements assigned to each processor. As a consequence, one would not want to call complex functions within `useric`, particularly any that would involve interelement (and, hence, interprocessor) communication. On the other hand, because `useric` is called only at the beginning of the computation, it's not imperative to optimize performance as might be the case for `userbc` (Sec. 6.2), which is called every timestep, for each field and for each gridpoint having user-prescribed boundary values.

One typically specifies initial conditions as a function of position. A simple example would be to give a tensor product of cosine functions in a square duct with $(x, y) \in [-1, 1]^2$ for arbitrary z. The following example illustrates this case:

```
c-------------------------------
c     I.C. EXAMPLE 1.
      subroutine useric(ix,iy,iz,eg)
      include 'SIZE'
      include 'NEKUSE'
      include 'TOTAL'

      integer e,eg

      ux=0
      uy=0
      uz=cos(.5*pi*x)*cos(.5*pi*y)
      temp=0

      return
      end
c-------------------------------
```

In this example, the independent variables, `x`, `y`, and `z`, as well as the dependent variables, `ux`, `uy`, `uz`, and `temp`, are passed through the common blocks in the `NEKUSE` include file. The constant `pi` comes through `TSTEP`, which is included in `TOTAL`. Initial conditions are set on a field-by-field basis, so the following example would be equivalent to the above.

```
c-------------------------------
c     I.C. EXAMPLE 2.
      subroutine useric(ix,iy,iz,eg)
      include 'SIZE'
      include 'NEKUSE'
      include 'TOTAL'

      integer e,eg

      if (ifield.eq.1) then      ! Set velocity i.c.
         ux=0
         uy=0
         uz=cos(.5*pi*x)*cos(.5*pi*y)
      elseif (ifield.eq.2) then  ! Set temperature i.c.
         temp=0
      else
         temp=0         ! Set i.c. for all other passive scalars
      endif

      return
      end
c-------------------------------
```

Notice that *all* passive scalars are set by defining `temp`. Different passive scalar assigments are discriminated by the value of `ifield`, which is set to 2 for temperature, 3 for the next passive scalar, etc. (Precise usage of `useric` can be found via `grep -i useric *.f` in `nek5_svn/trunk/nek`.)

6.2 Setting Boundary Conditions

Boundary conditions are assigned in a manner similar to initial conditions, save for a few technical differences. First, `userbc()` is called for every timestep, rather than just at the start of a simulation. Second, it is called only for a subset of the points in the domain, rather than for every point. In a parallel computation, it is likely that only a fraction of the processors will actually call `userbc` and one must therefore be concerned with load-balance as well as avoidance of any communication, for example, as would be required to resolve vector reduction operations such as global max (`glmax`) or a global sum (`glsum`). Because of the potential for load imbalance in high processor-count cases, it is common to precompute and store expensive function evaluations in `userchk()` and to then de-reference the results in `userbc`. The advantage of this approach is that it allows for vectorization (which is inhibited with the point-by-point calling convention of `userbc`) and/or storage of temporally-invariant results. The examples below illustrate some of these points.

In the first example, we see that the boundary condition assignment is identical to that of the initial condition above. The only difference in the argument list is the inclusion of `iside`, which indicates the element face number that is requiring boundary conditions. Generally, one does not need to reference `iside`, it is included simply to provide an additional boundary discriminator. (Another boundary condition discriminator is the 3-character string `cbu`, which is passed through `NEKUSE`.)

```
c-------------------------------
c     B.C. EXAMPLE 1.
      subroutine userbc(ix,iy,iz,iside,eg)
      include 'SIZE'
      include 'NEKUSE'
      include 'TOTAL'

      integer e,eg

      ux=0
      uy=0
      uz=cos(.5*pi*x)*cos(.5*pi*y)
      temp=0

      return
      end
c-------------------------------
```

The second example illustrates a case where the z-component of velocity is defined by the contents of an array that the user would typically declare and fill in `usrchk()`.

```
c-------------------------------
c     B.C. EXAMPLE 2. Illustrating dereference of
c                     a previously computed BC.
      subroutine userbc(ix,iy,iz,iside,eg)
      include 'SIZE'
      include 'NEKUSE'
      include 'TOTAL'

      common /my_uvw/ u3(lx1,ly1,lz1,lelt)

      integer e,eg

      e = gllel(eg)  ! local element number

      if (ifield.eq.1) then      ! Set velocity b.c.
         ux=0
         uy=0
         uz=u3(ix,iy,iz,e)
      elseif (ifield.eq.2) then  ! Set temperature b.c.
         temp=0
      else
         temp=0      ! Set b.c. for all other passive scalars
      endif

      return
      end
c-------------------------------
```

The code snippet below shows a typical way to fill `u3()` on the first call.

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

      common /my_uvw/ u3(lx1,ly1,lz1,lelt)

      n = nx1*ny1*nz1*nelv

      if (istep.eq.0) then
         do i=1,n
```

```
            x=xm1(i,1,1,1)
            y=ym1(i,1,1,1)
            u3(i,1,1,1)=cos(.5*pi*x)*cos(.5*pi*y)
         enddo
      endif

      return
      end
c--------------------------------
```

A time varying boundary condition can also be prescribed either by modifying `userchk` or directly in `userbc`, as illustrated in the next two examples.

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

      common /my_uvw/ u3(lx1,ly1,lz1,lelt)

      n = nx1*ny1*nz1*nelv

      eps   = 0.1
      omega = 5.0
      amp   = 1 + eps*sin(omega*time)

      do i=1,n
         x=xm1(i,1,1,1)
         y=ym1(i,1,1,1)
         u3(i,1,1,1)=cos(.5*pi*x)*cos(.5*pi*y)*amp
      enddo

      return
      end
c--------------------------------


c--------------------------------
c     B.C. EXAMPLE 3. - time varying boundary condition
      subroutine userbc(ix,iy,iz,iside,eg)
      include 'SIZE'
      include 'NEKUSE'
      include 'TOTAL'

      integer e,eg
      eps   = 0.1
      omega = 5.0
      amp   = 1 + eps*sin(omega*time)

      ux=0
      uy=0
      uz=amp*cos(.5*pi*x)*cos(.5*pi*y)

      temp=0

      return
      end
c-------------------------------
```

Chapter 7. Timestepping

In this chapter, we consider theoretical and practical issues relating to the timestepping schemes used in Nek5000. In particular, we constrast the conditionally-stable BDFk/EXTk scheme with the "unconditionally-stable" characteristics-based timestepper.

7.1 Theoretical Background

We consider semi-implicit numerical solution of convection-dominated problems such as the incompressible Navier-Stokes equations

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} &= -\nabla p + \frac{1}{Re}\nabla^2\mathbf{u} \quad \text{in } \Omega, \\ \nabla\cdot\mathbf{u} &= 0 \quad \text{in } \Omega, \end{aligned} \tag{7.1}$$

and the convection-diffusion equation

$$\frac{\partial \phi}{\partial t} + \mathbf{c}\cdot\nabla\phi = \frac{1}{Pe}\nabla^2\phi + f, \qquad \text{in } \Omega. \tag{7.2}$$

subject to appropriate initial and boundary conditions on $\mathbf{u}$ and ϕ. As with the Navier-Stokes case, we assume a divergence-free convecting velocity field, $\nabla\cdot\mathbf{c} = 0$.

We would like to construct a high-order scheme in time that treats the nonsymmetric convection term explicitly while treating the diffusion term implicitly. One possible approach is to use Crank-Nicolson (CN) for the diffusion term and third-order Adams-Bashforth (AB3) for the convection term. This approach amounts to computing ϕ^n by approximately integrating the time-derivative of ϕ over the interval $[t^{n-1}, t^n]$. The diffusion term is integrated using the trapezoidal rule, and the convection term is integrated by extrapolating the convection contribution over the interval $[t^{n-1}, t^n]$ based on available values from prior timesteps, t^{n-q}, $q = 1, \ldots, 3$. The CN/AB3 scheme is

$$\begin{aligned} \frac{\phi^n - \phi^{n-1}}{\Delta t} &= \frac{1}{2}\frac{1}{Pe}(\nabla^2\phi^n + \nabla^2\phi^{n-1}) \\ &+ \frac{1}{12}(23\,\mathbf{c}\cdot\nabla\phi|_{t^{n-1}} - 16\,\mathbf{c}\cdot\nabla\phi|_{t^{n-2}} + 5\,\mathbf{c}\cdot\nabla\phi|_{t^{n-3}}) + \frac{1}{2}(f^n + f^{n-1} \end{aligned} \quad (7.3$$

If f happens to be dependent on ϕ^n, it can alternatively be lumped with the AB3 terms. The scheme is globally second-order accurate in time. We choose AB3 over AB2 because the work is essentially the same and because the stability region for AB3 encloses a significant portion of the imaginary axis and thus contains the eigenvalues of the convection-diffusion operator in the zero-diffusion limit.

An alternative approach to the CN-AB3 scheme is to use backward differentiation of order k (BDFk). Here the time-derivative of ϕ is approximated to order k, using a one-sided difference formula, and the convection and diffusion terms are evaluated at time t^n. To avoid implicit treatment of the nonsymmetric convection term, we approximate $\mathbf{c} \cdot \nabla\phi$ at t^n by extrapolation of order k (EXTk), using values from t^{n-q}, $q = 1, \ldots, k$. The values for f can be extrapolated, if they are dependent on ϕ^n, or evaluated directly at t^n, if known. A BDF2/EXT2 formulation is given by

$$\frac{3\phi^n - 4\phi^{n-1} + \phi^{n-2}}{2\Delta t} = \frac{1}{Pe}\nabla^2\phi^n - \left(2\,\mathbf{c} \cdot \nabla\phi|_{t^{n-1}} - \mathbf{c} \cdot \nabla\phi|_{t^{n-2}}\right) + f^n \tag{7.4}$$

The scheme is globally second-order accurate in time. It is readily extended to kth-order accuracy by changing the second-order approximation to the time derivative on the left-hand side of (7.4) and using kth-order extrapolation for the convection term on the right. As with the CN/AB3 scheme, the cost for this approach is dominated by the work associated with the implicit solve. The added cost of increasing k is thus essentially nil. Note that it's not appropriate to use ABk coefficients on the right-hand side of (7.4) because (7.4) is an approximation to the *value* of the function at t^n, rather than an approximation to the *integral* of the function over the interval $[t^{n-1}, t^n]$.

7.2 Generalization of BDFk/EXTk

To extend the the BDFk/EXTk scheme to variable timestep sizes, we first rewrite (7.4) in a more general form:

$$\sum_{q=0}^{k} \beta_q \phi^{n-q} = -\sum_{p=1}^{k} \gamma_p\, \mathbf{c} \cdot \nabla\phi|_{t^{n-p}} + \mathcal{I}^n, \tag{7.5}$$

where $\mathcal{I}^n$ accounts for all the implicit contributions. The terms β_q are the differentiation coefficients associated with the approximation of a derivative at time t^n based on known values at t^{n-q}, $q = 0, \ldots, k$. The terms γ_p are interpolation coefficients required to approximate a function at time t^n based on known values at

t^{n-p}, $p = 1, \ldots, k$. These coefficients are readily determined from any standard approach to polynomial interpolation at knot points $[t^{n-k}, ..., t^{n-j}]$ ($j = 0$ or 1). A convenient code is the general-purpose routine for interpolation/differentiation weights, FD_WEIGHTS, developed by Bengt Fornberg [?]. Note that, because of potential round-off errors, it is probably better for long time integrations to use just the *change* in time values, $\delta^{n-q} := t^{n-q} - t^n$, rather than the time values themselves, to determine the interpolation/differentiation coefficients. Of course, the values of δ^{n-q} should be computed by summing Δt^{n-j}, $j = 0, \ldots, q$, rather than taking the difference $t^{n-q} - t^n$.

7.3 Stability Considerations

The explicit treatment of the convection term leads to timestep size restrictions in the CN/AB3 and BDFk/EXTk schemes given in (7.3)–(7.5) above. Assuming a closed domain ($\mathbf{c} \cdot \mathbf{n} \equiv 0$), the linear operator will be skew-symmetric in the zero-diffusion limit, Assuming that the skew symmetry of the continuous (in space) operator is preserved in the numerical discretization, we should seek a temporal discretization that encloses a portion of the imaginary axis near the origin. Both AB3 and BDF3/EXT3 accomplish this.

Consider the linear convection problem

$$\frac{\partial \phi}{\partial t} + \mathbf{c} \cdot \nabla \phi = 0, \tag{7.6}$$

subject to appropriate initial and boundary conditions, and assume that it has been discretized in space to yield

$$\frac{d\underline{\phi}}{dt} = -C\underline{\phi}, \tag{7.7}$$

where $\underline{\phi}$ is the vector of unknown basis coefficients and C is the discrete skew-symmetric convection operator. Discretizing (7.7) in time using AB3 yields

$$\underline{\phi}^n \;=\; \underline{\phi}^{n-1} \;-\; \frac{\Delta t}{12}(23C\underline{\phi}^{n-1} - 16C\underline{\phi}^{n-2} + 5C\underline{\phi}^{n-3}), \tag{7.8}$$

where we have assumed that the timestep size, Δt, is constant and that the velocity field $\mathbf{c}$ and, hence, C, is time invariant. The stability region for AB3 is found by considering the model problem $u_t = \lambda u$, inserting this into (7.8) ($\underline{\phi} \leftarrow u$, $-C \leftarrow \lambda$), and solving for the locus of points in the complex $(\lambda\Delta t)$-plane such that the magnitude of u is unchanging (see, e.g., [?] and Appendix A.) Figure 7.1

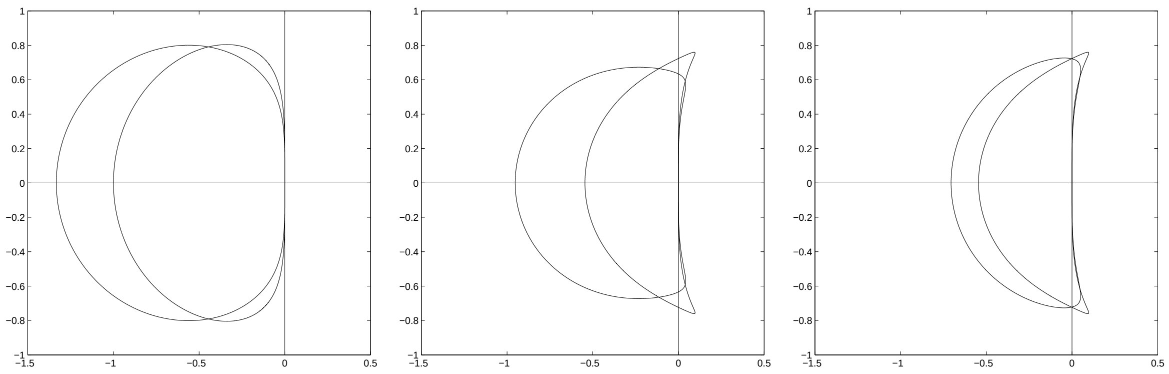

Figure 7.1: Stability regions for (left) AB2 and BDF2/EXT2, (center) AB3 and BDF3/EXT3, and (right) AB3 and BDF2/EXT2a.

shows the stability region for AB3 and for BDF3/EXT3. We see that both schemes enclose a substantial portion of the imaginary axis near the origin. It is also possible to improve the stability of BDF2/EXT2 by using a three-term treatment of the second-order extrapolation. For example, for fixed Δt, the choice $\gamma_1 = 8/3$, $\gamma_2 = -7/3$, and $\gamma_3 = 2/3$, will provide a stability region that is similar to AB3, as seen in the right panel of Fig. 7.1.

The implications of Fig. 7.1 are that, for stability, one must choose $\Delta t \leq z_{\text{crit}} / \max|\lambda|$, where $z_{\text{crit}} = .7236$ for AB3 and .6339 for BDF3/EXT3. The value of $\max|\lambda|$ depends on the *spatial* discretization and on the computational mesh size. For classical centered finite differences (FD), one has

$$\max|\lambda| = \frac{|\mathbf{c}|}{\Delta x},$$

which leads to the well known Courant-Friedrichs-Lewy (CFL) number

$$CFL := \frac{|\mathbf{c}|\Delta t}{\Delta x} = \max|\lambda|\Delta t.$$

Thus, for FD + AB3, we must have $CFL \leq .7236$. For a spatial discretizations based on one-dimensional Fourier methods, the maximum eigenvalue is

$$\max|\lambda| = \pi\frac{|\mathbf{c}|}{\Delta x}$$

[**?**], which implies that $CFL \leq .7236/\pi$ is required for stability. For spectral element methods, the bound is

$$\max|\lambda| = S\frac{|\mathbf{c}|}{\Delta x_{\min}},$$

where S is a value ranging between 1.52 and 1.16 as the polynomial order N is increased from 3 to ∞ [**?**]. Note that, for the spectral element method, $\Delta x_{\min}$ scales as $O(N^{-2})$ as a result of the clustering of the Gauss-Lobatto-Legendre points near the ends of the reference interval. Schemes that avoid the CFL restriction are thus of greater interest in the SEM case.

7.4 Characteristics Method

The characteristics scheme due to Pironneau [**?**] avoids the CFL constraint in the semi-implicit formulation by rewriting the convective term in (7.2) as a material derivative to yield

$$\frac{D\phi}{Dt} = \frac{1}{Pe}\nabla^2\phi + f, \tag{7.9}$$

where

$$\frac{D\phi}{Dt} := \frac{\partial\phi}{\partial t} + \mathbf{c}\cdot\nabla\phi \tag{7.10}$$

represents the change of ϕ along the path line (or characteristic) associated with the convecting velocity field $\mathbf{c}$. The basic idea behind this method is to apply BDFk to (7.9), to yield (e.g., for k=2):

$$\frac{3\phi^n - 4\tilde{\phi}^{n-1} + \tilde{\phi}^{n-2}}{2\Delta t} = \frac{1}{Pe}\nabla^2\phi^n + f^n. \tag{7.11}$$

(As before, if f^n is dependent upon ϕ^n, it can be approximated using EXTk.) In (7.11), $\tilde{\phi}^{n-q}$ represents the value of ϕ at an earlier point in time (t^{n-q}) and at the point in space, $\mathbf{X}^{n-q}$, from which it originates prior to being convected forward by the velocity field. That is,

$$\tilde{\phi}^{n-q}(\mathbf{x}) := \phi(\mathbf{X}^{n-q}(\mathbf{x}), t^{n-q}). \tag{7.12}$$

Note that the domain of $\tilde{\phi}^{n-q}$ is coincident with $\Omega(t^n)$, whereas the domain of $\phi(., t^{n-q})$ must be coincident with $\Omega(t^{n-q})$. This is a subtle point, but the implications are that spatial discretization of (7.11) can proceed with the usual weighted residual formalism, that is, by multiplying (7.11) by a test function v, integrating over Ω, and restricting the search (trial) and test spaces to finite dimensional subspaces of $H^1_0(\Omega)$.

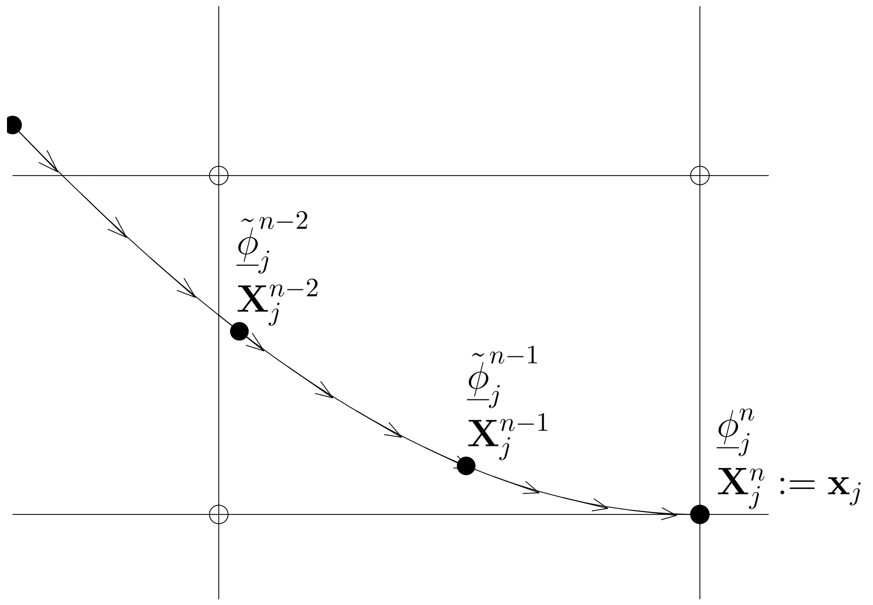
$\underline{\tilde{\phi}}_j^{n-2}$
$\mathbf{X}_j^{n-2}$
$\underline{\tilde{\phi}}_j^{n-1}$
$\mathbf{X}_j^{n-1}$
$\underline{\phi}_j^{n}$
$\mathbf{X}_j^{n} := \mathbf{x}_j$

Figure 7.2: BDF2 approximation of material derivative $D\phi/Dt$ along the characteristic emanating from $\mathbf{x}_j$. $\mathbf{X}_j^{n-q}$ is the foot of the characteristic corresponding to $t^n - q\Delta t$.

To illustrate the principal implementation aspects of the characteristics scheme, we consider the fully discretized system (in time and space), assuming a nodal basis for ϕ and introducing gridpoints $\mathbf{x}_j$. Assume that the spatial discretization of (7.11) leads to the following linear system

$$\left(-\frac{1}{Pe}A + \frac{3}{2\Delta t}B\right)\underline{\phi}^n \;=\; B\underline{g}^n, \tag{7.13}$$

where A is the stiffness matrix, B is the mass matrix, and the right-hand side

$$\underline{g}^n \;:=\; B\left(\underline{f}^n + \frac{4}{2\Delta t}\underline{\tilde{\phi}}^{n-1} - \frac{1}{2\Delta t}\underline{\tilde{\phi}}^{n-2}\right), \tag{7.14}$$

accounts for the data and the values of ϕ at the feet of the characteristics.

The central question with the characteristics scheme is how to find the values of ϕ at the foot of the characteristic associated with gridpoint $\mathbf{x}_j$, that is,

$$\tilde{\phi}_j^{n-q} := \phi(\mathbf{X}_j^{n-q}, t^{n-q}),$$

where $\mathbf{X}_j^{n-q}$ is the foot of the characteristic emanating from $\mathbf{x}_j$, as illustrated in Fig. 7.2. The standard semi-Lagrangian formulation, suggested by Pironneau and followed by others, is,

- starting at each point $\mathbf{x}_j$, march backwards along the velocity characteristic for an amount of time $q\Delta t$,
- evaluate $\mathbf{X}_j^{n-q}$ (the endpoint of the trajectory in the preceding step)
- then evaluate $\phi(\mathbf{X}^{n-q}, t^{n-q}) := \phi^{n-q}(\mathbf{X}^{n-q})$.

This obviously requires that one interpolate $\phi^{n-q}(\mathbf{x})$ at points that are not coincident with the gridpoints since, in general, one will not have $\mathbf{X}_j^{n-q} = \mathbf{x}_{j'}$ for any j'. In fact, the situation is somewhat worse. In addition to interpolating ϕ, it is necessary to interpolate the vector field $\mathbf{c}$ at each point along the characteristic, in order to accurately march back along the trajectory. In low-order unstructured methods, most of the interpolation cost arises from determination of the cell containing the point of interest. In high-order methods, one has an additional cost because off-grid interpolation for an Nth-order discretization in d space dimensions has a complexity of $O(N^d)$ *per interpolated value.* For a spectral element discretization comprising E elements of order N, there are $\sim EN^d$ gridpoints, and the semi-Lagrangian evaluation cost is thus $O(EN^{2d})$. Moreover, the constant is not small, given that several interpolations are required, the Lagrangian bases need to be constructed, and, for deformed geometries, Newton's method must be applied to find the computational coordinates in the reference element as a function of $\mathbf{X}_j^{n-q}$. This cost must be contrasted to the cost of Eulerian operator evaluations which have operation count scaling as $O(EN^{d+1})$ and relatively small constants (e.g., for the spectral element method, one can evaluate $w = \mathbf{c} \cdot \nabla\phi$ in approximately $2dEN^{d+1}$ operations).

7.5 The Operator-Integration Factor Scheme

The operator-integration factor scheme (OIFS), due to Maday, Patera, and Rønquist [?] can be viewed as a variant of the characteristics scheme (7.11) in which the values of $\tilde{\phi}^{n-q}$ are obtained without using off-grid interpolation. The basic idea is to recognize that to evaluate $\tilde{\phi}^{n-q}$, one does not need to know $\mathbf{X}^{n-q}$ as (7.12) would seem to indicate. Rather, one can find $\tilde{\phi}^{n-q}$ by simply solving the following initial-boundary value problem:

$$\frac{\partial\tilde{\phi}}{\partial s} + \mathbf{c}\cdot\nabla\tilde{\phi} \;=\; 0, \qquad s \in [t^{n-q}, t^n] \tag{7.15}$$
$$\tilde{\phi}(\mathbf{x}, t^{n-q}) \;=\; \phi(\mathbf{x}, t^{n-q}) \qquad\qquad \tilde{\phi}(\mathbf{x}, t) \;=\; \phi(\mathbf{x}, t)\ \forall\mathbf{x} \,\in\, \partial\Omega_{\mathbf{c}}.$$

Here, $\partial\Omega_{\mathbf{c}}$ is defined as the subset of the boundary $\partial\Omega$ where $\mathbf{c}\cdot\mathbf{n} < 0$, that is, the portion of $\partial\Omega$ having incoming velocity characteristics. (Note that, in practice, one can assign $\tilde{\phi}(\mathbf{x},t) = \phi(\mathbf{x},t^{n-q})$ on $\partial\Omega_{\mathbf{c}}$. PF, verify.)

Equation (7.15) is a pure convection problem. It has the effect of propagating the initial condition (in this case, the values of ϕ at time t^{n-q}) forward, along the characteristics of the convecting field $\mathbf{c}$. After a time of $q\Delta t$, the values of $\tilde{\phi}$ at the gridpoints $\mathbf{x}_j$ are precisely the desired values $\tilde{\phi}^{n-q}(\mathbf{x}_j) := \phi(\mathbf{X}_j^{n-q}, t^{n-q})$. These values can be inserted directly into (7.14). Note that *no* interpolation is required to find $\tilde{\phi}^{n-q}$ because (7.15) is solved in the Eulerian framework, using explicit time marching with a stepsize, $\Delta s \leq \Delta t$, that is sufficiently small to satisfy the CFL constraint. Because of the short duration of the time integration ($q\Delta t$), it is preferable to solve (7.15) using a multistage integrator such as Runge-Kutta, rather than a multistep integrator such as ABk, to avoid the start-up problems associated with lack of information prior to the first time step. Note that if $\mathbf{c}$ is a function of t and known only at discrete times t^{n-q}, then interpolation will need to be used to approximate it at intermediate times (e.g., $t^{n-q}+\Delta s$, $t^{n-q}+2\Delta s$, etc.). Fornberg's interpolation routine is also useful in this context.

In summary, the OIF scheme involves solving (7.15) k times, with different initial conditions and over different time intervals; summing these solutions together with weights β_q to construct the right-hand side (7.14); and finally solving the Helmholtz problem (7.13) to advance the solution to the next step.

As it stands, the OIF scheme has a complexity scaling as $O(k^2)$, for there are k integrations to be performed over time intervals ranging from Δt to $k\Delta t$ in length. For example, suppse that one wishes to use OIFS with BDF3, using a Δt-based CFL of 5. To compute $\tilde{\phi}^{n-3}$, one needs to integrate (7.15) from t^{n-3} to t^n, or a total of $3\Delta t$ in time. Using RK4, whose stability curve intersects the imaginary axis at $\lambda\Delta t \approx 2i$, and assuming, with some margin of safety, that the maximum eigenvalue of the spectral element discretization satifes $\lambda_{\max}\Delta t \approx 2CFL$, one should set the substep size $\Delta s \approx \Delta t/5$. That is, five RK4 steps, each requiring four function evaluations, would be required for *each* Δt interval covered by the integration. The total number of function evaluations would thus be $5 \times 4 \times 3 = 60$ for the computation of $\tilde{\phi}^{n-3}$. For $\tilde{\phi}^{n-2}$, one would have 40, and for $\tilde{\phi}^{n-1}$, one would have 20.

This complexity can be reduced to $O(k)$ by recognizing that (7.15) is *linear*. Therefore superposition can be used to compute the k

solutions in a single pass. We replace (7.15) with the following system: For $q = k, k-1, \dots, 1$:

$$\begin{aligned} \frac{\partial \psi^q}{\partial s} + \mathbf{c} \cdot \nabla \psi^q &= 0, \qquad s \in [t^{n-q}, t^{n-q+1}] \qquad (7.16) \\ \psi(\mathbf{x}, t^{n-q}) &= \psi^{q+1}(t^{n-q}) + \beta_q \phi(\mathbf{x}, t^{n-q}) \qquad \psi(\mathbf{x}, t) = \phi(\mathbf{x}, t) \; \forall \mathbf{x} \in \partial\Omega_{\mathbf{c}}. \end{aligned}$$

With $\psi^{k+1} := 0$, the final result is

$$\psi^1(t^n) = \sum_{q=1}^{k} \beta_q \tilde{\phi}^{n-q}, \qquad (7.17)$$

which is the desired contribution to the right-hand side of (7.14).

7.6 Extension to Navier-Stokes

The extension of the characteristics/OIF method to the Navier-Stokes equations is straightforward. Following the formalism used above for the convection-diffusion equations, we write (7.1) as

$$\begin{aligned} \frac{Du}{Dt} &= -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u} \quad \text{in } \Omega, \qquad (7.18) \\ \nabla \cdot \mathbf{u} &= 0 \quad \text{in } \Omega. \end{aligned}$$

The (here nonlinear) convective term has again been expressed as a material derivative. As in the convection-diffusion problem, the objective is to replace the conditionally stable treatment of the convective term with an unconditionally stable BDFk treatment of the material derivative. (Note that large values of k are of particular interest here since one needs to compensate for accuracy loss resulting from larger values of Δt.) Using this approach, the semidiscretized Navier-Stokes equations for $k = 2$ are

$$\begin{aligned} \frac{3\mathbf{u}^n - 4\tilde{\mathbf{u}}^{n-1} + \tilde{\mathbf{u}}^{n-2}}{2\Delta t} = &= -\nabla p^n + \frac{1}{Re} \nabla^2 \mathbf{u}^n \quad \text{in } \Omega, (7.19) \\ \nabla \cdot \mathbf{u}^n &= 0 \quad \text{in } \Omega. \end{aligned}$$

Again, the values $\tilde{\mathbf{u}}^{n-q}$ represent the field $\mathbf{u}$ at the foot of the characteristics. These may be computed using the usual semi-Lagrangian formulation involving off-grid interpolation, or by the OIF approach, in which one solves the convective subproblems

$$\begin{aligned} \frac{\partial \tilde{\mathbf{u}}}{\partial s} + \mathbf{u} \cdot \nabla \tilde{\mathbf{u}} &= 0, \qquad s \in [t^{n-q}, t^n] \qquad (7.20) \\ \tilde{\mathbf{u}}(\mathbf{x}, t^{n-q}) &= \phi(\mathbf{x}, t^{n-q}) \qquad \tilde{\mathbf{u}}(\mathbf{x}, t) = \phi(\mathbf{x}, t) \; \forall \mathbf{x} \in \partial\Omega_{\mathbf{c}}, \end{aligned}$$

where, $\partial\Omega_{\mathbf{c}}$ is defined as the subset of the boundary $\partial\Omega$ where $\mathbf{u}\cdot\mathbf{n} < 0$.

Note that there should be no confusion in (7.20) between the *convected* field, $\tilde{\mathbf{u}}$, and the *convecting* field, $\mathbf{u}$, which is the solution to (7.19). In solving (7.20), one treats $\tilde{\mathbf{u}}$ as a passive advected vector field. Values of $\mathbf{u}$ on the interval $[t^{n-k}, t^n]$ are computed by interpolating from the known fields $\mathbf{u}^{n-k}$, $\mathbf{u}^{n-k+1}, \ldots, \mathbf{u}^{n-1}$, which guarantees a divergence-free convecting field in (7.20). Unlike the convection-diffusion problem, however, one does not have access to $\mathbf{u}^n$, so $\mathbf{u}$ is effectively *extrapolated* on $(t^{n-1}, t]$. One has the option of iterating between (7.20) and (7.19) to get an estimate of $\mathbf{u}^n$, which can then be employed in the interpolation step required when solving (7.20) on a second (third, etc.) pass. Formally, this should not be required, as the extrapolation-based scheme is $O(\Delta t^k)$ accurate. However, it may be of interest in particularly challenging applications. If properly coded, (e.g., by relying on general interpolation routines such as Fornberg's), this iterative approach can be included as an option in the OIFS implementation from the outset with little overhead.

Remark on Steady State Error

We comment that one advantage of the BDFk/EXTk formulation is that there is no temporal error for problems that have evolved to a steady state solution. That is, the final result (and error) is independent of Δt. To see this, consider (7.4) under steady-state conditions, $\phi^n = \phi^{n-1} = \ldots$. The lefthand side of (7.4) vanishes identically and represents the only term containing Δt. The right-hand side vanishes through a balance of the diffusion and advection terms that are independent of Δt and the error will (ultimately) reflect only the spatial discretization parameters.

On the other hand, the OIFS formulation *does* retain a steady state error that is influenced by Δt. This is evident from (7.11). Under steady state conditions, we again have $\phi^n = \phi^{n-1} = \ldots$, but these are not the terms that are required to balance. The lefthand side of (7.11) inolves weighted differences of ϕ^n, $\tilde{\phi}^{n-1}$, and $\tilde{\phi}^{n-2}$ that are divided by Δt. In fact, there should be no expectation that the lefthand side should vanish, even as $\Delta t \longrightarrow 0$, because the lefthand side represents the variation of ϕ as one travels along the characteristic. This material derivative will generally be nonzero and the temporal finite difference scheme will retain an $O(\Delta t)$ dependence even under steady-state conditions. The characteristics-based OIFS scheme amounts to mixing spatial and temporal discretizations and care must be exercised when implementing such methods, particularly in a high-order context.

Appendix A: Stability Region Evaluation

The matlab code used to generate Fig. 7.1 is presented below. The neutral stability curves are generated by considering application of a particular timestepping scheme to the homogeneous model problem

$$\frac{du}{dt} = \lambda u.$$

Assume that a uniform stepsize, Δt, is employed and that the timestepping scheme is of the form:

$$\frac{1}{\Delta t}\sum_{q=0}^{k_1} \beta_q u^{n-q} \quad = \quad \lambda \sum_{p=1}^{k_2} \gamma_p u^{n-p}. \tag{7.21}$$

Eq. (7.21) has solutions of the form

$$u^n = (z)^n u^0, \tag{7.22}$$

where (.) is used to indicate that the (generally complex) argument z is raised to the nth power. In order for (7.22) to be neutrally stable, we require $|z| = 1$, that is, $z = e^{i\theta}$ for some value of θ between 0 and 2π. Inserting (7.22) into (7.21) and solving for $\lambda\Delta t$ yields

$$\lambda \Delta t = \frac{\sum_{q=0}^{k_1} \beta_q e^{-iq\theta}}{\sum_{p=1}^{k_2} \gamma_p e^{-ip\theta}.}. \tag{7.23}$$

Evaluating (7.23) for $0 \leq \theta < 2\pi$ yields the locus associated with the neutral stability curve.

The matlab code below fills a vector `ei` with complex values on the unit circle, then creates column vectors `exp(q*ei)` for `q=1,0,-1,-2`. Linear combinations of these column vectors form the numerator and denominator of (7.23). The matlab `plot` routine is used to plot the locus of complex pairs.

Matlab Code for Stability Analysis

```
ii=sqrt(-1);
th=0:.001:2*pi;  th=th';
ith=ii*th;
ei=exp(ith);
E = [ ei 1+0*ei 1./ei 1./(ei.*ei) ];

ep = 1.e-13

ab0   = [1 0.0 0.0 0.]';
ab1   = [0 1.0 0.0 0.]';
ab2   = [0 1.5 -.5 0.]';
ab3   = [0 23./12. -16./12. 5./12.]';
bdf1  = (([ 1.  -1.  0.  0.])/1.)';
bdf2  = (([ 3.  -4.  1.  0.])/2.)';
bdf3  = (([11. -18.  9. -2.])/6.)';
ex0   = [0 1  0 0]';
ex1   = [0 2 -1 0]';
ex2   = [0 3 -3 1]';
yaxis = [-1.0*ii 1.0*ii]';
xaxis = [-2.0+ep*ii 2.0+ep*ii]';
du    = [1. -1. 0. 0. ]';

hold off; plot (yaxis,'k-'); axis square; axis([-1.5 0.5 -1 1]); hold on;
                                                   plot (xaxis,'k-');
  ab3       = (E*du)./(E*ab3);                     plot (ab3 ,'r-');
  bdf3ex2   = (E*bdf3)./(E*ex2);                   plot (bdf3ex2,'k-');
  print -deps ab3bdf3.eps

hold off; plot (yaxis,'k-'); axis square; axis([-1.5 0.5 -1 1]); hold on;
                                                   plot (xaxis,'k-');
  ab2       = (E*du)./(E*ab2);                     plot (ab2 ,'r-');
  bdf2ex1   = (E*bdf2)./(E*ex1);                   plot (bdf2ex1,'k-');
  print -deps ab2bdf2.eps
```