
NekIBM Documentation

Release beta

Yunchao Yang

Jun 11, 2020

CONTENTS

NEKIBM'S DOCUMENTATION

1.1 Quickstart

1.1.1 Directory structure

1.1.2 Case files

1.1.3 Scripts

Let's walk through some useful batch scripts:

- `makenek <case>` compiles your case
- `nek/nekb <case>` runs a serial job in foreground or background
- `nekmpi/nekbmpi <case> <number of ranks>` runs a parallel job
- `neknec <case1> <cas2> <ranks 1> <ranks 2>` runs Nek5000 with two overlapping component grids
- `visnek <case>` creates metadata file required by [VisIt](#) and [ParaView](#).
- `mvn <old name> <new name>` renames all case files
- `cpn <old name> <new name>` copies all case files

1.1.4 Meshing

Nek5000 is mainly a solver. However, simple box type meshes can be generated with the `genbox` tool. For more complex meshes please consider using `PRENEK` and the meshing tools `nekmerge` and `n2to3`. We provide mesh converters like `exo2nek` and `msh2nek` which are quite handy if you want to use your favorite mesh generator. Also check our [Bazaar](#) for 3rd party meshing tools.

1.1.5 Visualization

Nek5000 output (`.fld` or `0.f%05d`) files can be read by [VisIt](#) or [ParaView](#). This requires using `visnek` to generate a metadata file. There is also an build-in X-Window based postprocessor called `POSTNEK` located in `tools`.

1.2 Tutorials

The following tutorials are provided to familiarize beginners with the features and typical workflow of Nek5000. For a user brand new to Nek5000, we strongly recommend beginning with the Periodic Hill tutorial.

1.3 Theory

1.3.1 Computational Approach

The spatial discretization is based on the spectral element method (SEM) [Patera1984], 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 N th-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 [Orszag1980]. 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 [Denville2002] for more details.

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-walls`. Additionally Nek5000 can handle conjugate heat transfer problems.

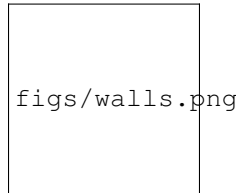


Fig. 1: Computational domain showing respective fluid and solid subdomains, Ω_f and Ω_s . The shared boundaries are denoted $\partial\Omega_f = \partial\Omega_s$ and the solid boundary which is not shared by fluid is $\overline{\partial\Omega_s}$, while the fluid boundary not shared by solid $\overline{\partial\Omega_f}$.

1.3.2 Incompressible Navier-Stokes Equations

The governing equations of flow motion in dimensional form are

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \rho \mathbf{f}, \text{ in } \Omega_f, \quad (\text{Momentum}) \quad (1.1)$$

where $\boldsymbol{\tau} = \mu[\nabla \mathbf{u} + \nabla \mathbf{u}^T]$ and \mathbf{f} is a user defined acceleration.

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

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

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

1.3.3 Non-Dimensional Navier-Stokes

Let us introduce the following non-dimensional variables $\mathbf{x}^* = \frac{\mathbf{x}}{L}$, $\mathbf{u}^* = \frac{\mathbf{u}}{U}$, $t^* = \frac{tU}{L}$, and $\mathbf{f}^* = \frac{\mathbf{f}L}{U^2}$. 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:

$$\frac{\partial \mathbf{u}^*}{\partial t^*} + \mathbf{u}^* \cdot \nabla \mathbf{u}^* = -\nabla p^* + \frac{1}{Re} \nabla \cdot \boldsymbol{\tau}^* + \mathbf{f}^*. \quad (1.3)$$

where $\boldsymbol{\tau}^* = [\nabla \mathbf{u}^* + \nabla \mathbf{u}^{*T}]$ and \mathbf{f}^* is the dimensionless user defined forcing function, e.g. gravity.

The non-dimensional number here is the Reynolds number $Re = \frac{\rho U L}{\mu}$.

1.3.4 Energy Equation

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

$$\rho c_p \left(\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T \right) = \nabla \cdot (k \nabla T) + q_{vol}, \text{ in } \Omega_f \cup \Omega_s \text{ (Energy)} \quad (1.4)$$

1.3.5 Non-Dimensional Energy / Passive Scalar Equation

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

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

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

1.3.6 Passive Scalars

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

$$(\rho c_p)_i \left(\frac{\partial \phi_i}{\partial t} + \mathbf{u} \cdot \nabla \phi_i \right) = \nabla \cdot (k_i \nabla \phi_i) + (q_{vol})_i. \quad (1.6)$$

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.

1.3.7 Unsteady Stokes

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

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} \right) = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \rho \mathbf{f}, \text{ in } \Omega_f \text{ (Momentum)} \quad (1.7)$$

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

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

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.

1.3.8 Steady Stokes

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

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

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

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

1.3.9 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*

$$\begin{aligned} \rho \left(\frac{\partial \mathbf{u}'_i}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}'_i + \mathbf{u}'_i \cdot \nabla \mathbf{u} \right) &= -\nabla p'_i + \mu \nabla^2 \mathbf{u}'_i \\ \nabla \cdot \mathbf{u}'_i &= 0 \end{aligned}$$

for multiple perturbation fields $i = 1, 2, \dots$ 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.

1.3.10 Steady Conduction

The energy equation (??) in which the advection term $\mathbf{u} \cdot \nabla T$ and the transient term $\partial T / \partial t$ are zero. In essence this represents a Poisson equation.

1.3.11 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. $\rho = f(p, T)$. Nek5000 includes the ability to solve the low-Mach approximation of the compressible Navier-Stokes, $\rho \approx f(T)$. 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 are

$$\begin{aligned} \rho \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) &= -\nabla p + \nabla \cdot \boldsymbol{\tau} + \rho \mathbf{f} \\ \nabla \cdot \mathbf{u} &= -\frac{1}{\rho} \frac{d\rho}{dT} \left(\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T \right) \\ \rho c_p \left(\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T \right) &= -\nabla \cdot k \nabla T + q_{vol} \end{aligned} \quad (1.11)$$

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

This allows for both variable density and variable viscosity. The system is solved by substituting $\rho \approx f(T)$ into the continuity equation and obtaining a so-called thermal divergence.

1.3.12 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

$$\begin{aligned} \rho \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) &= -\nabla p + \mu \Delta \mathbf{u} + \mathbf{B} \cdot \nabla \mathbf{B}, \\ \nabla \cdot \mathbf{u} &= 0 \\ \frac{\partial \mathbf{B}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{B} &= -\nabla q + \eta \Delta \mathbf{B} + \mathbf{B} \cdot \nabla \mathbf{u}, \\ \nabla \cdot \mathbf{B} &= 0 \end{aligned}$$

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 ($\mathbf{z}^\pm = \mathbf{u} \pm \mathbf{b}$)

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

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

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

1.3.13 Arbitrary Lagrangian-Eulerian (ALE)

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

$$\begin{aligned}\frac{\partial \mathbf{u}}{\partial t} &= -\nabla p + \frac{1}{Re} \nabla \cdot (\nabla + \nabla^T) \mathbf{u} + NL, \\ \nabla \cdot \mathbf{u} &= 0\end{aligned}\tag{1.12}$$

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 [Ho1989]. 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 Eq. (??): Find $(\mathbf{u}, p) \in X^N \times Y^N$ such that:

$$\frac{d}{dt}(\mathbf{v}, \mathbf{u}) = (\nabla \cdot \mathbf{v}, p) - \frac{2}{Re}(\nabla \mathbf{v}, \mathbf{S}) + (\mathbf{v}, NL) + c(\mathbf{v}, \mathbf{w}, \mathbf{u}), \quad (\nabla \cdot \mathbf{u}, q) = 0,\tag{1.13}$$

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 [Maday1989], (\cdot, \cdot) 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 Eq. (??) is the trilinear form involving the mesh velocity

$$c(\mathbf{v}, \mathbf{w}, \mathbf{u}) := \int_{\Omega(t)} \sum_{i=1}^3 \sum_{j=1}^3 v_i \frac{\partial w_j u_i}{\partial x_j} dV,\tag{1.14}$$

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 Eq. (??) 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 Eq. (??) is

$$\frac{1}{2\Delta t} [3(\mathbf{v}^n, \mathbf{u}^n)^n - 4(\mathbf{v}^{n-1}, \mathbf{u}^{n-1})^{n-1} + (\mathbf{v}^{n-2}, \mathbf{u}^{n-2})^{n-2}] = L^n(\mathbf{u}) + 2\widetilde{NL}^{n-1} - \widetilde{NL}^{n-2}.\tag{1.15}$$

Here, $L^n(\mathbf{u})$ accounts for all *linear* terms in Eq. (??), 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 (??), at time-level t^{n-q} . The superscript on the inner-products $(\cdot, \cdot)^{n-q}$ 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.

1.4 Problem Setup

A complete problem setup in Nek5000 requires a set of case files which are used for defining the geometry and physics of a problem. Details of how to setup a case in Nek5000 are outlined in the following sections:

1.5 Appendices

1.5.1 Build Options

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:

Table 1: Compiler options

name	values	default	description
PPLIST	string		list of pre-processor symbols (CVODE, ...)
MPI	1, 0	1	use MPI (needed for a multiprocessor computation)
FC	string	optional	Fortran compiler (mpif77)
CC	string	optional	C compiler (mpicc)
FCLAGS	string	optional	optional Fortran compilation flags
CCLAGS	string	optional	optional C compilation flags
SOURCE_ROOT	string	optional	path of Nek5000 source
USR	string	optional	object list of additional files to compile make instructions (<code>makefile_usr.inc</code> required)
USR_LFLAGS	string	optional	optional linking flags
PROFILING	1, 0	1	enable internal timers for performance statistics
VISIT	1, 0	0	Toggles Visit in situ. See <code>Visit_in_situ</code> for details
VISIT_INSTALL	string	VISIT in situ	Path to VISIT install path. See <code>Visit_in_situ</code> for details.
VISIT_STOP	true, false	false	When running VISIT in situ, simulation stops after step 1 to connect VISIT.

The `PPLIST` field can be used to activate several features at compilation time. A list of possible options is below:

Table 2: PPLIST options

Symbol	Description
NOMPIIO	deactivate MPI-IO support
BGQ	use Blue Gene Q optimized mxm
XSMM	use libxsmm for mxm
CVODE	compile with CVODE support for scalars
VENDOR_BLAS	use VENDOR BLAS/LAPACK
EXTBAR	add underscore to exit call (for BGQ)
NEKNEK	activate overlapping mesh solver (experimental)
CMTNEK	activate discontinuous Galerkin compressible-flow solver (experimental)

In addition to these preprocessor items, the user can add compilation and linking flags. `FFLAGS` allows the user to add Fortran compilation flags while `CCFLAGS` allows the user to add C compilation flags. These will be compiler dependent and the user is encouraged to consult the manual of the compiler if specific options are needed/desired. A commonly used flag is `-mmodel` 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`.

1.5.2 Internal Input Parameters/Switches

Parameters

P001 density for the case of constant properties

P002 dynamic viscosity

P007 heat capacity for the case of constant properties

P008 conductivity for the case of constant properties

P010 simulation end time

P011 number of time steps

P012 time step size

P014 time frequency to dump fld files

P015 step frequency to dump fld files

P021 pressure solver tolerance

P022 velocity solver tolerance

P023 number of passive scalars

P024 relative tolerance for Helmholtz solver

P025 absolute tolerance for Helmholtz solver

P026 target Courant number (determines the number of RK4 substeps for OIFS)

P027 temporal discretization order

P028 temporal discretization order for mesh solver

P029 magnetic viscosity

P030 material properties (0: constant, 1: uservp)

P031 number of perturbation modes in linearized N-S.

- P032** number of boundary conditions in .re2 file
- P033** first field index in .re2
- P040** pressure coarse grid solver (0: XXT, 1: AMG)
- P041** 1 \rightarrow multiplicative SEMG
- P042** linear solver for the pressure equation (0: GMRES, 1: CG)
- P043** 0: additive multilevel scheme - 1: original two level scheme.
- P044** 0=E-based additive Schwarz for PnPn-2; 1=A-based.
- P045** Free-surface stability control (defaults to 1.0)
- P046** if > 0 , do not set Initial Condition (no call to subroutine SETICS).
- P047** Poisson ratio for mesh elasticity solve (default 0.4)
- P054** direction of fixed flowrate (1: x, 2: y, 3: z), negative means fixed bulk
- P055** volumetric flowrate or bulk velocity (see p054) for periodic case
- P059** deformed element switch
- P060** initialize velocity to 1e-10 (for steady Stokes problem).
- P062** byte swap for output
- P063** output precision (4: SP, 8: DP)
- P064** restart perturbation solution
- P065** number of I/O nodes (if < 0 write in separate subdirectories).
- P066** output format (0: ASCII, 4: legacy binary, 6: binary)
- P067** read format
- P068** averaging frequency in `avg_all` (0: every timestep).
- P084** custom initial time step
- P086** use skew-symmetric instead of convective form.
- P093** number of previous solutions to use for residual projection.

P094 number of steps starting residual projection for velocity and passive scalars

P095 number of steps starting residual projection for pressure

P099 dealiasing mode (< 0 : disabled, 3: old dealiasing, 4: new dealiasing)

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

P101 number of additional modes to filter

P103 filter weight for last mode

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

P116 NELX number of elements in x for FTP

P117 NELY number of elements in y for FTP

P118 NELZ number of elements in z for FTP

Logical switches

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

IFLOMACH use low Mach number formulation

IFMGRID moving grid

IFMVBD moving boundary (for free surface flow)

IFCHAR use characteristics for convection operator

IFSYNC use upfront synchronization

IFUSERVP user-defined properties

Other Input Variables

filterType 0: explicit, 1: HPT-RT

restol(:) field solver tolerance

1.5.3 Commonly used Variables

Solution Variables

Variable Name	Size	Type	Short Description
vx	(lx1,ly1,lz1,lev)	real	x-velocity (u)
vy	(lx1,ly1,lz1,lev)	real	y-velocity (v)
vz	(lx1,ly1,lz1,lev)	real	z-velocity (w)
pr	(lx2,ly2,lz2,lev)	real	pressure (pr)
t	(lx1,ly1,lz1,lelt,ldimt)	real	temperature (t) and passive scalars (ps)
vtrans	(lx1,ly1,lz1,lelt,ldimt+1)	real	convective coefficient
vdiff	(lx1,ly1,lz1,lelt,ldimt+1)	real	diffusion coefficient
vxlag	(lx1,ly1,lz1,lev,2)	real	x-velocity at previous time steps
vylag	(lx1,ly1,lz1,lev,2)	real	y-velocity at previous time steps
vzlag	(lx1,ly1,lz1,lev,2)	real	z-velocity at previous time steps
prlag	(lx2,ly2,lz2,lev,lorder2)	real	pressure at previous time steps
tlag	(lx1,ly1,lz1,lev,lorder-1,ldimt+1)	real	t and ps at previous time steps
time	–	real	physical time
dt	–	real	time step size
dtlag	(10)	real	previous time step sizes
istep	–	integer	time step number

Geometry Variables

Variable Name	Size	Type	Short Description
xm1	(lx1,ly1,lz1,lelt)	real	x-coordinates for velocity mesh
ym1	(lx1,ly1,lz1,lelt)	real	y-coordinates for velocity mesh
zm1	(lx1,ly1,lz1,lelt)	real	z-coordinates for velocity mesh
bm1	(lx1,ly1,lz1,lelt)	real	mass matrix for velocity mesh
binvm1	(lx1,ly1,lz1,lev)	real	inverse mass matrix for velocity mesh
bintm1	(lx1,ly1,lz1,lelt)	real	inverse mass matrix for t mesh
volvm1	–	real	total volume for velocity mesh
voltm1	–	real	total volume for t mesh
xm2	(lx2,ly2,lz2,lev)	real	x-coordinates for pressure mesh
ym2	(lx2,ly2,lz2,lev)	real	y-coordinates for pressure mesh
zm2	(lx2,ly2,lz2,lev)	real	z-coordinates for pressure mesh
unx	(lx1,ly1,6,lelt)	real	x-component of face unit normal
uny	(lx1,ly1,6,lelt)	real	y-component of face unit normal
unz	(lx1,ly1,6,lelt)	real	z-component of face unit normal
area	(lx1,ly1,6,lelt)	real	face area (surface integral weights)

Problem Setup Variables

Variable Name	Size	Type	Short Description
nid	–	integer	MPI rank id (lowest rank is always 0)
nio	–	integer	I/O node id
nelv	–	integer	number of elements in velocity mesh
nelt	–	integer	number of elements in t mesh
ndim	–	integer	dimensionality of problem (i.e. 2 or 3)
nsteps	–	integer	number of time steps to run
iostep	–	integer	time steps between data output
cbc	(6,lelt,ldimt+1)	character*3	boundary condition
lglel	(lelt)	integer	local to global element number map
gllel	(lelg)	integer	global to local element number map

Averaging Variables

Arrays associated with the `avg_all` subroutine

Variable Name	Size	Type	Short Description
uavg	(ax1,ay1,az1,lelt)	real	time averaged x-velocity
vavg	(ax1,ay1,az1,lelt)	real	time averaged y-velocity
wavg	(ax1,ay1,az1,lelt)	real	time averaged z-velocity
pavg	(ax2,ay2,az2,lelt)	real	time averaged pressure
tavg	(ax1,ay1,az1,lelt,ldimt)	real	time averaged temperature and passive scalars
urms	(ax1,ay1,az1,lelt)	real	time averaged u^2
vrms	(ax1,ay1,az1,lelt)	real	time averaged v^2
wrms	(ax1,ay1,az1,lelt)	real	time averaged w^2
prms	(ax1,ay1,az1,lelt)	real	time averaged pr^2
trms	(ax1,ay1,az1,lelt,ldimt)	real	time averaged t^2 and ps^2
uvms	(ax1,ay1,az1,lelt)	real	time averaged uv
vwms	(ax1,ay1,az1,lelt)	real	time averaged vw
wums	(ax1,ay1,az1,lelt)	real	time averaged wu
iastep	–	integer	time steps between averaged data output

1.5.4 Commonly used 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 `l`, 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`.

`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 (a, b, c, n)`

`function glsum (a, n)`

Computes the global sum of the real arrays `a`, with number of local entries `n`

`function iglsum (a, n)`

Computes the global sum of the integer arrays `a`, with number of local entries `n`

`function i8glsum (a, n)`

Computes the global sum of the integer*8 arrays `a`, with number of local entries `n`

subroutine surface_int(dphi,dS,phi,ielem,side) Computes the surface integral of scalar array `phi` over face `side` of element `ielem`. The resulting integral is stored in `dphi` and the area in `dS`.

1.5.5 Generating a Mesh with Genbox

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:mesh_axi1`. 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)
```

figs/mesh_axi1.png

Fig. 2: Axisymmetric pipe mesh.

- The first line of this file supplies the name of an existing 2D `.rea` file that has the appropriate run parameters (viscosity, timestep size, etc.). These parameters can be modified later, but it is important that `axisymmetric.rea` be a 2D file, and not a 3D file.
- The second line indicates the number of fields for this simulation, in this case, just 1, corresponding to the velocity field (i.e., no heat transfer).
- The next set of lines just shows how one can place comments into a `genbox` input file.
- The line that starts with “Box” indicates that a new box is starting, and that the following lines describe a typical box input. Other possible key characters (the first character of Box, “B”) are “C” and “M”, more on those later.
- The first line after “Box” specifies the number of elements in the x and y directions. The fact that these values are negative indicates that you want `genbox` to automatically generate the element distribution along each axis, rather than providing it by hand. (More on this below.)
- The next line specifies the 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.

- 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 $u = 0$.

Graded Mesh

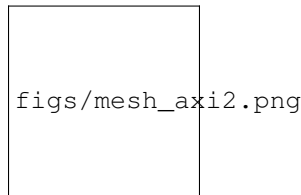


Fig. 3: Axisymmetric pipe mesh, graded

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

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

to

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

yields the mesh shown in `fig:mesh_axi2`.

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		

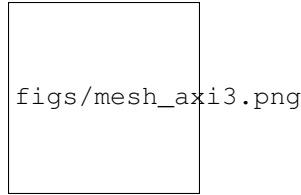


Fig. 4: Axisymmetric pipe mesh, user specified.

Here, the positive number of elements for the y direction indicates that `genbox` is expecting $N_{ely}+1$ values of y positions on the y -element distribution line. This is the `genbox` default, which explains why it corresponds to $N_{ely} > 0$. The corresponding mesh is shown in `fig:mesh_axi3`.

1.5.6 Mesh Modification

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 $y \rightarrow 0$, 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)
end do

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. The result of above changes is shown in `fig:wavypipe`.



figs/wavypipe.png

Fig. 5: Axisymmetric pipe mesh.

Cylindrical/Cartesian-transition Annuli



figs/cylbox_2d.png

Fig. 6: Cylinder mesh



figs/cylbox_2da.png

Fig. 7: Cylinder mesh

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`. `fig:cylbox_2d` 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:cylbox_2d`. The mesh has been quad-refined once with oct-refine option of `prenek`. The 3D counterpart to this mesh could be joined to a hemisphere/Cartesian transition built with the spherical mesh option in `prenek`.

1.5.7 Mesh Extrusion and Mirroring

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,O):
v
Enter Z (6) boundary condition (v,O):
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-walls`. 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-walls`. 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 imported meshes.

1.6 Bibliography

Recommended Citation

Zwick, D. (2019). ppiclF: A Parallel Particle-In-Cell Library in Fortran. *Journal of Open Source Software*. 4(37), 1400. <https://doi.org/10.21105/joss.01400>

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

BIBLIOGRAPHY

- [Denville2002] Deville, M.O. and P.F. Fischer and E.H. Mund. *High-order methods for incompressible fluid flow*. Cambridge University Press, 2002.
- [Ho1989] Ho, L.W. A Legendre spectral element method for simulation of incompressible unsteady viscous free-surface flows. Ph.D. thesis, Massachusetts Institute of Technology, 1989.
- [Maday1989] Maday, Y. and A.T. Patera. Spectral element methods for the Navier-Stokes equations. In *State-of-the-Art Surveys in Computational Mechanics*, pages 71–143. ASME, New York, 1989.
- [Orszag1980] Orszag, S.A. “Spectral methods for problems in complex geometry.” *J. Comput. Phys.* **37**:70–92, 1980.
- [Patera1984] Patera, A.T. “A spectral element method for fluid dynamics : laminar flow in a channel expansion.” *J. Comput. Phys.* **54**:468–488, 1984.
- [Mellen2000] Mellen, C. P., Fröhlich, J., Rodi, W. “Large-eddy simulation of the flow over periodic hills.” *16th IMACS World Congress*, Lausanne, Switzerland.