CMT-nek

Generated by Doxygen 1.8.5

Sun Aug 30 2020 18:00:29

Contents

Chapter 1

The discontinuous Galerkin spectral element method

CMT-nek is an implementation of the **discontinuous Galerkin spectral element method (DGSEM)** written for systems of conservation laws. General descriptions and theory of DG methods is found in a few textbooks. I recommend **??**, where much of the notation in this document is taken. It solves for 5 conserved variables $\mathbf{U}(\mathbf{x},t) \in \mathbb{R}^5$, $\mathbf{x} = (x_1,x_2,x_3)^\top \in \Omega \subset \mathbb{R}^3$, $t \in \mathbb{R}^+$, satisfying the conservation law

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{H} = \mathbf{R}, \tag{1.1}$$

where each of the 5 equations has its own flux vector $\mathbf{H}(\mathbf{U}) : \mathbf{U} \to \mathbb{R}^3$.

1.1 The discontinuous Galerkin spectral element method (DGSEM)

CMT-nek solves Equation 1.1 by partitioning the domain $\Omega \subset \mathbb{R}^d$, $d = 1,2,3$ into nelt non-overlapping elements (Figure 1.1), the e^{th} of which is Ω_e, and marching the left-hand-side of Equation 1.1 forward in time on each element. The right-hand-side of Equation 1.1 is discretized on each element in a very peculiar way called **the two-point form of DGSEM** that is only briefly summarized here. The two-point form of DGSEM brings discontinuous Galerkin methods and finite difference methods together in a very peculiar way, and the reader is strongly encouraged to study the bibliography carefully.

1.1.1 Weighted residuals

Galerkin methods force an inner product of Equation 1.1 with a test function to vanish for every value of the test function in some basis spanning a finite-dimensional space of test functions χ. Integrating this inner product by parts on a given elementΩ_e gives us the **weak form** of the discontinuous Galerkin weighted residual statement for the governing equation 1.1:

$$\int_{\Omega_e} v(\mathbf{x}) \frac{\partial \mathbf{U}(\mathbf{x})}{\partial t} dV = \int_{\Omega_e} (\nabla v) \cdot \mathbf{H}\, dV - \int_{\partial \Omega_e} v(\mathbf{x}) \mathbf{H}^*(\mathbf{U}^-,\mathbf{U}^+) \cdot \hat{\mathbf{n}} dA + \int_{\Omega_e} \mathbf{R}(\mathbf{x}) v(\mathbf{x}) dV, \tag{1.2}$$

where the surface integral term $\mathbf{H} \cdot \hat{\mathbf{n}}$ in the surface integral has been replaced by the **numerical flux** $\mathbf{H}^*(\mathbf{U}^-,\mathbf{U}^+) \cdot \hat{\mathbf{n}}$ that, since χ is a broken space defined on each element, resolves the discontinuities between the representation of $\mathbf{U}$ on the faces of Ω_e and the corresponding $\mathbf{U}$ in the neighbors sharing faces $f \in \partial \Omega_e$. That is,

$$U^-(\mathbf{x}) \equiv U(\mathbf{x}) \text{ taken from the } \textbf{interior}\text{, or trace, of } \Omega_e \tag{1.3}$$

$$U^+(\mathbf{x}) \equiv U(\mathbf{x}) \text{ taken from the } \textbf{element adjacent to } \Omega_e \text{ sharing } \partial\Omega_e. \tag{1.4}$$

The numerical flux functon $\mathbf{H}^*$ is critically important to the stability and convergence properties of DGSEM, and will be presented in more detail in §**??**.

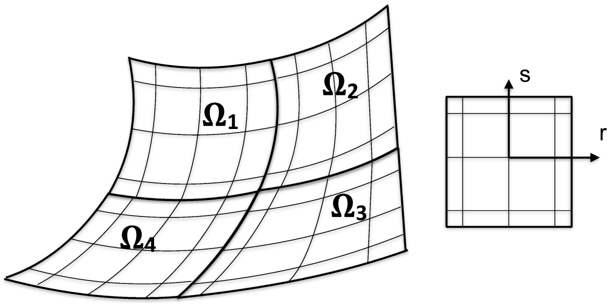


Figure 1.1: A schematic representation of spectral elements and the reference element in two dimensions.

Figure 1.1 also illustrates how a point $\mathbf{x} \in \Omega_e$ is isoparametrically mapped to $\mathbf{r} \equiv (r,s,t)^\top = (r_1,r_2,r_3)^\top$ in the reference element $\hat{\Omega} = [-1,1]^3$, This transformation has, $\forall \mathbf{x} \in \Omega_e$, Jacobian $J(\mathbf{x}) \equiv |\partial \mathbf{x}/\partial \mathbf{r}|$ and metrics $\partial r_i/\partial x_j$.

We integrate the right-hand-side (RHS) of Equation 1.2 by parts a second time to get the **"strong" form** of the weighted residual statement,

$$\int_{\Omega_e} v(\mathbf{x}) \frac{\partial \mathbf{U}(\mathbf{x})}{\partial t} dV = \int_{\Omega_e} v \nabla \cdot \mathbf{H} dV - \int_{\partial \Omega_e} v(\mathbf{x}) \left(\mathbf{H} - \mathbf{H}^*\right) \cdot \hat{\mathbf{n}} dA + \int_{\Omega_e} \mathbf{R}(\mathbf{x}) v(\mathbf{x}) dV. \tag{1.5}$$

The distinction between weak and strong forms is ultimately unimportant in DGSEM, but strong form (Equation 1.5) is clearer and more convenient.

1.1.2 Quadrature and approximation

All integrals in Equation 1.5 are now approximated by **Gaussian quadrature**, as described in §whatever of **?**. We evaluate the solution $\mathbf{U}$ and various functions of it (like fluxes $\mathbf{H}$ and source terms $\mathbf{R}$) on a grid of N^3 **Gaussian quadrature nodes** within each element. This means:

1. The approximation space χ becomes $[\mathbb{P}^{N-1}]^3$, a Cartesian product of the space of all polynomials of degree $N-1$.
2. The basis functions are a **nested tensor product** of the **interpolating Lagrange polynomials** evaluated on N^2 lines of N Gaussian quadrature nodes. This is described in more detail in §of **?**.
3. The discrete unknowns are the **nodal values** $\mathbf{U}(x_i, y_j, z_k)$ at each of the N^3 Gaussian nodes in each element.

Finally, within the family of Gaussian quadrature we specifically choose the **Gauss-Legendre-Lobatto (GLL)** nodes. Formulas for ω and $\mathbf{r}$ may be found in Appendix of **?**.

Nodal values at grid points within a given element are arranged into vectors lexicographically:

$$\mathbf{u} \equiv \begin{bmatrix} U(x_1,y_1,z_1) \\ U(x_2,y_1,z_1) \\ \vdots \\ U(x_N,y_N,z_N) \end{bmatrix}, \mathbf{v} \equiv \begin{bmatrix} v(x_1,y_1,z_1) \\ v(x_2,y_1,z_1) \\ \vdots \\ v(x_N,y_N,z_N) \end{bmatrix}, \mathbf{h}_1 \equiv \begin{bmatrix} H_x(U(x_1,y_1,z_1)) \\ H_x(U(x_2,y_1,z_1)) \\ \vdots \\ H_x(U(x_N,y_N,z_N)) \end{bmatrix}, \tag{1.6}$$

with $u_l = U(x_i,y_j,z_k)$ when $l = i + N(j-1) + N^2(k-1)$.

These vectors may be catenated one element after the other into a vector $\underline{\mathbf{u}}_L$ of N^3K nodal values for the quadrature nodes in the entire mesh.

To assist notation for scalar multiplication, we introduce the diagonal matrix formed from an arbitrary nodal vector $\mathbf{f}$

$$\text{diag}(\mathbf{f}) \equiv \begin{bmatrix} f(x_1,y_1,z_1) & & 0 \\ & \ddots & \\ & f(x_i,y_j,z_k) & \\ & \ddots & \\ 0 & & f(x_N,y_N,z_N) \end{bmatrix}. \tag{1.7}$$

Consider the left-hand-side of Equation 1.5. Approximating everything with the nodal representation on N^3 GLL points described above, means our discrete equation becomes, at the top level,

$$\int_{\Omega_e} v(\mathbf{x})\frac{\partial \mathbf{U}(\mathbf{x})}{\partial t}dV \approx \mathbf{v}^\top \mathbf{B}\frac{\partial \mathbf{u}}{\partial t} = \text{RHS}, \tag{1.8}$$

where RHS is the right-hand-side of Equation 1.5 evaluated at each GLL node. Bold-faced quantities are nodal vectors (Equation 1.6) including the mass matrix $\mathbf{B}$,

$$\mathbf{B} \equiv \begin{bmatrix} \ddots & & 0 \\ & \omega_i\omega_j\omega_k J(\mathbf{x}(r_i,s_j,t_k)) & \\ 0 & & \ddots \end{bmatrix}, \tag{1.9}$$

where ω_i is the GLL quadrature weight associated with the i^{th}quadrature node, and $J_e(\mathbf{x})$ is the Jacobian J of the mesh transformation described above at the GLL points within Ω_e.

The Galerkin statement is enforced for all possible test functions in χ by equating coefficients of the test function $\mathbf{v}$ on both sides of Equation 1.8. Further multiplication of Equation 1.8 by $\mathbf{B}^{-1}$ produces the semidiscrete method of lines

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{B}^{-1}\text{RHS} = I_{\text{vol}} - I_{\text{sfc}}, \tag{1.10}$$

which may be integrated in time to solve for the nodal values $\mathbf{u}$ of the conserved variables. Right-hand-side terms in Equation 1.10 are essentially the effects contributions of individual GLL nodes to integrals (hence the "I" abbreviation). We describe those in the following section.

1.2 Split forms of evaluating summation-by-parts operators

We're solving a differential equation. We need to take derivatives of things like U and $\mathbf{H}$ approximated by polynomials on each element. Finite differences are appropriate for computing derivatives of interpolating polynomials. Algorithms are commonplace for finite differences on each of N points expressed as products between a $N \times N$ differentiation matrix $\mathscr{D}$ and a vector of N nodal values in one spatial dimension,

$$\mathscr{D}\mathbf{v} \approx \left[\left.\frac{dv}{dr}\right|_{r_1}, \cdots, \left.\frac{dv}{dr}\right|_{r_N} \right]^\top. \tag{1.11}$$

$\mathscr{D}$ is computed in Nek5000 using formulas derived in (**?**, §).

Derivatives of U for all N^3 GLL nodes in Ω_e in each of the three coordinate directions on $\hat{\Omega}$ are evaluated via the triple Kronecker product (see (**?**, §)) of $\mathscr{D}$ with $\mathbf{I} \in \mathbb{R}^{N\times N}$:

$$\mathbf{D}_r = \mathbf{D}_{r_1} = \mathbf{I}\otimes\mathbf{I}\otimes\mathscr{D}, \quad \mathbf{D}_s = \mathbf{D}_{r_2} = \mathbf{I}\otimes\mathscr{D}\otimes\mathbf{I}, \quad \mathbf{D}_t = \mathbf{D}_{r_3} = \mathscr{D}\otimes\mathbf{I}\otimes\mathbf{I}. \tag{1.12}$$

On the GLL nodes, $\mathscr{D}$ has the **summation-by-parts property**:

$$(\mathbf{B}\mathscr{D})^\top + \mathbf{B}\mathscr{D} = \text{diag}\left([-1,0,\ldots,0,1]\right), \tag{1.13}$$

which means that integration by parts in inner products like Equations 1.2 and 1.5 is done *exactly* at the discrete level *even if* the underlying quadrature is not itself exact. Naïvely, we would write I_{vol} in Equation 1.10 by approximating the volume integral[1]

$$\sum_{e=1}^{nelt}\int_{\Omega_e} v\nabla\cdot\mathbf{H}\,dV \approx \mathbf{v}^\top\mathbf{B}\mathbf{D}_i\left[\text{diag}\left(\frac{\partial r_i}{\partial x_j}\right)\mathbf{h}_j\right], \tag{1.14}$$

where $\mathbf{h}_j$ is the flux in the x_j direction at the GLL nodes on all elements. We would then equate coefficients of $\mathbf{v}$ and left multiply by $\mathbf{B}^{-1}$ to get I_{vol}. However, it turns out**?** Equation 1.14 is itself further approximated by **a subcell flux difference**. Considering a single GLL grid line on an undeformed element (such that $r = x_1$ for brevity, Fisher**?** proved[2]

$$(\mathbf{D}_r\mathbf{h}_r)_{(ijk)} \approx \frac{F_{(i+1,jk)} - F_{(ijk)}}{\omega_i} \tag{1.15}$$

[1] The Einstein summation convention applies to multiplication of derivatives in the direction i on the GLL grid with flux in the physical direction j.

[2] Cartesian indices in parentheses like "(i)" refer to the i^{th} element of an array and are *not* subject to summation convention.

to within the truncation error of the Lagrange polynomials on GLL nodes with nodal values of fluxes $\mathbf{h}$. Fisher introduced a **auxiliary flux function** F that provided a way of enforcing bounds on fluxes *even in the face of quadrature errors in evaluating them.* The importance of this for stabilization will be discussed in §.

Finally, Fisher derived a **two-point split form** for Equation 1.15 that allowed (further) approximation of F with *yet another* flux function $F^{\#}$

$$\frac{F_{(i+1,jk)} - F_{(ijk)}}{\omega_i} \approx 2\sum_{l=1}^{N} \mathscr{D}_{il} F^{\#}\left(\mathbf{u}_{(ijk)}, \mathbf{u}_{(ljk)}\right), \tag{1.16}$$

where $F^{\#}\left(\mathbf{U}_a, \mathbf{U}_b\right)$ is a flux function of two arguments instead of one! At the very least, it must be consistent with the physical flux function.

$$\mathbf{F}^{\#}\left(\mathbf{U},\mathbf{U}\right) = \mathbf{H}(\mathbf{U}) \tag{1.17}$$

and symmetric in its two arguments.

So, to summarize, Equation 1.16 is, for system-dependent choices of $\mathbf{F}^{\#}$ to be shown in §, a *stabilizing* way of approximately evaluating the volume integral in Equation 1.5 for schemes based on summation-by-parts (SBP) operators. Finite differences on GLL nodes in DGSEM are SBP operators. While Equation 1.16 disrupts the matrix-vector product $\mathbf{Du}$, it retains N^4 scaling in higher dimensions and promises some unique stability properties on top of conservation and high-order accuracy**?**. As a notational shorthand, we abbreviate Equation 1.16 as

$$\mathbb{D}(F^{\#}) \equiv 2\sum_{l=1}^{N} \mathscr{D}_{il} F^{\#}\left(\mathbf{u}_{(ijk)}, \mathbf{u}_{(ljk)}\right), \tag{1.18}$$

where $F^{\#}\left(\mathbf{U}_a, \mathbf{U}_b\right)$ is a flux function of two arguments instead of one! At the very least, it must be consistent with the physical flux function for a

Gassner, Winters and Kopriva**?** (and the works cited therein) derive and explore several properties and features of these "**two-point split forms**" for the Euler equations. They transform them to deformed spectral elements as follows. Let indices *i,j,k,l* denote individual GLL nodes within $\underline{\mathbf{u}}$ in three-dimensional storage or matrix elements. First, fluxes must be transformed in a freestream-preserving way into the **contravariant frame** aligned along the GLL grid within a given element. Again, we subscript $\mathbf{H} = (\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3)$ by the physical-space direction $j \in [1,3]$. Again, we distinguish Cartesian tensor indices from GLL nodes and matrix elements by writing, for example, the x_2-direction flux at the i^{th} GLL node as $H_2\left(\mathbf{U}_{(i)}\right)$ and *demanding that summation convention apply* to the index on H but *NOT* to the index on $\mathbf{U}$!!! Then, for a given conserved variable, the integrand in the volume integral in Equation 1.5 becomes

$$[\mathbf{D}_r\mathbf{h}]_{(ijk)} \approx \left[\mathbb{D}_r\left(F^{\#}\right)\right]_{(ijk)} \equiv 2\sum_{l=1}^{N} \mathscr{D}_{(il)} \left\{\left\{J\text{diag}\left(\frac{\partial r_1}{\partial x_k}\right)\right\}\right\}_{((i,l)jk)} F_k^{\#}\left(\mathbf{U}_{(ijk)}, \mathbf{U}_{(ljk)}\right), \tag{1.19}$$

$$[\mathbf{D}_s\mathbf{h}]_{(ijk)} \approx \left[\mathbb{D}_s\left(F^{\#}\right)\right]_{(ijk)} \equiv 2\sum_{l=1}^{N} \mathscr{D}_{(jl)} \left\{\left\{J\text{diag}\left(\frac{\partial r_2}{\partial x_k}\right)\right\}\right\}_{(i(j,l)k)} F_k^{\#}\left(\mathbf{U}_{(ijk)}, \mathbf{U}_{(ilk)}\right), \tag{1.20}$$

$$[\mathbf{D}_t\mathbf{h}]_{(ijk)} \approx \left[\mathbb{D}_t\left(F^{\#}\right)\right]_{(ijk)} \equiv 2\sum_{l=1}^{N} \mathscr{D}_{(kl)} \left\{\left\{J\text{diag}\left(\frac{\partial r_3}{\partial x_k}\right)\right\}\right\}_{(ij(k,l))} F_k^{\#}\left(\mathbf{U}_{(ijk)}, \mathbf{U}_{(ijl)}\right), \tag{1.21}$$

where we have introduced notation for an **average** between two grid points along a line of fixed r in the grid on the reference element,

$$\{\{U\}\}_{((i,l)jk)} \equiv \frac{1}{2}\left(U_{(ijk)} + U_{(ljk)}\right), \tag{1.22}$$

$$\{\{U\}\}_{(i(j,l)k)} \equiv \frac{1}{2}\left(U_{(ijk)} + U_{(ilk)}\right), \tag{1.23}$$

$$\{\{U\}\}_{(ij(k,l))} \equiv \frac{1}{2}\left(U_{(ijk)} + U_{(ijl)}\right). \tag{1.24}$$

These averages will be needed to define various flux functions of interest too. Equations 1.19 through 1.21 are applied component-wise to each of the conserved variables in $\mathbf{H}$ (with correponding components in $\mathbf{F}$).

So, we have an elaborate way of writing the discrete approximation to the volume integral on the right-hand-side of Equation 1.10:

$$I_{\text{vol}} \equiv \text{diag}\left(\frac{1}{J}\right)\sum_{j=1}^{3} \mathbb{D}_j\mathbf{h} \tag{1.25}$$

where $\mathbb{D}_j$ is defined by Equations 1.19 through 1.21.

1.2.1 A short note on strong form

The extra discontinuous surface flux in Equation 1.5 is evaluated on the fly by modifying the differentiation matrix $\mathscr{D}$. In the code,

$$dstrong = \mathscr{D} - 2\text{diag}\left([-1/\omega_1, 0, \ldots, 0, 1/\omega_N]\right)\mathbb{D} \tag{1.26}$$

is enough to include the surface integral from integrating by parts twice. dstrong is evaluated in chainrule_metrics subroutine of intpdiff.

1.3 Surface integral terms

$$I_{\text{sfc}} = -\left(\sum_{f=1}^{6} \mathbf{E}^{(f)\top}\mathbf{B}^{(f)}\left[\text{diag}\left(\hat{\mathbf{n}}_j^{(f)}\right)\left(\mathbf{h}_j - \mathbf{h}_j^{*(f)}\right)\right]\right). \tag{1.27}$$

where the "face" mass matrix $\mathbf{B}^{(f)}$ is

$$\mathbf{B}^{(f)} \equiv \begin{bmatrix} \ddots & & 0 \\ & \omega_i\omega_j J^{(f)}(\mathbf{x}(r_i, s_j)) & \\ 0 & & \ddots \end{bmatrix} \in \mathbb{R}^{N^2\times N^2}, \tag{1.28}$$

and we note that the vectors of nodal values $\hat{\mathbf{n}}_j^{(f)}$ and $\mathbf{h}_j^{(f)}$ in 1.27 have only N^2 elements since they correspond to GLL nodes on face f. Crucially, we have also introduced the *restriction operator* $\mathbf{E}^{(f)} \in \mathbb{R}^{N^2\times N^3}$ for the f^{th} face in 1.27. The restriction operator is defined as an indicator that is zero for all GLL nodes except those on the element faces $\partial\Omega_e$, an operation easily expressed as Kronecker products (for example, in the r_1−direction for $f = 1$ at $r = r_1 = -1$)

$$\mathbf{E}^{(1)} = \mathbf{I}\otimes\mathbf{I}\otimes\mathscr{E}_1 \tag{1.29}$$

of the $\mathbf{I} \in \mathbb{R}^{N\times N}$ identity matrix with unit vectors $\mathscr{E} \in \mathbb{R}^N$ (in 1.29, $\mathscr{E}_1 \equiv (1, 0, \ldots, 0)^\top$). Similar indicators may be built up in the r_2- and r_3-directions by following the ordering of Kronecker products in 1.12. Applying the matrix identity $(\mathbf{AB})^\top = \mathbf{B}^\top\mathbf{A}^\top$ to Equation 1.27 discretely represents the surface integral of the numerical flux on $\hat{\Omega}$:

$$\sum_{f=1}^{6}\iint_{-1}^{1} v(\mathbf{r})H_j^{*(f)}\hat{n}_j^{(f)}J^{(f)}\,dr\,ds = \sum_{f=1}^{6}\mathbf{v}^\top\mathbf{E}^{(f)\top}\mathbf{B}^{(f)}\left(\left[\text{diag}\left(\hat{\mathbf{n}}_j^{(f)}\right)\mathbf{h}_j^{*(f)}\right]\right). \tag{1.30}$$

1.3.1 Surface flux functions

According to a vast body of literature not cited here, summation-by-parts (SBP) operators need appropriate boundary treatment to behave well. These are called **simultaneous approximation terms** (SAT), and the family of methods that have nice conservation and stability properties always have these two things together and are called "SBP-SAT" methods. For the purposes of DGSEM, Gassner, Winters & Kopriva (, Eq) recommend a numerical flux function consisting of a symmetric function and an extra dissipative term:

$$\mathbf{H}^*\left(\mathbf{U}^-, \mathbf{U}^+\right) = \mathbf{F}^\#\left(\mathbf{U}^-, \mathbf{U}^+\right) + \mathbf{F}_{\text{stab}}\left(\mathbf{U}^-, \mathbf{U}^+\right), \tag{1.31}$$

where $\mathbf{H}^*$ is the flux between neighboring elements in Equation 1.5. Gassner, Winters and Kopriva have indeed recycled the same split-form flux function used in Equations 1.19 through 1.21, but instead of evaluating it at two separate points, they evaluate it using the two values present at each interface between elements.

SAT tend to be dissipative, and the dissipative nature of $\mathbf{H}^*$ in DG is needed in DGSEM as well. $\mathbf{F}^\#$ is augmented with a stabilizing flux $\mathbf{F}_{\text{stab}}$. Examples of this will be given in §.

Chapter 2

Time-marching and top-level assembly

A parallel program like CMT-nek only fits **nelt** elements on the memory available to a single MPI task. **lelt** is the maximum number of elements that may be stored on a single MPI task. It is set in the SIZE file. Quantities such as Equation 1.6 and 1.7 are stored in multidimensional arrays to facilitate nested-tensor-product operations. An example is the storage of the values of the x-coordinate at grid points in each element in the mesh that lies on a given MPI task.

Note on lz1: The triple "lx1,ly1,lz1" is mandated for all declarations and loops intended to cover or otherwise refer to the grid of N^d quadrature nodes within an element. Although lx1, ly1 and lz1 are not arbitrary, the limited rules surrounding their use does make it easier to reuse code and share it between 2D and 3D cases. An important rule for such sharing is in Table 2.2.

Table 2.3 shows where quadrature-related quantities live (again, in core/, not core/cmt/)

2.1 Data structure of surface quantities

Surface terms and nodal vectors of surface quantities are stored in a different format from the one introduced in Table 3.2 for general storage of variables on the whole mesh. Every face point in every element, from an element-centered point of view (i.e., every element has is own copy of values at nodes lying on its 2*ldim faces), fits in an array of size

```
nfq=lx1*lz1*2*ldim*nelt
```

ordered according to:

```
dimension(lx1,lz1,2*ldim,lelt)
```

I call this a "pile of faces." I also made a common block to store many of them, /CMTSURFLX/, that I had intended to be somewhat malleable. Thus, quantities in it are very large 1D arrays, indexed by whole-number multiples of nfq, and passed to different subroutines. **ONLY inside subroutines** are **dummy arguments** dimensioned (lx1,lz1,2*ldim,lelt). Every face has lx1$\times$lz1 nodes on the face. This makes it suitable for both 2D (lz1=1) and 3D

Mathematical variable	variable in code	common	include file
N	lx1=ly1		SIZE
dimension d	ldim		SIZE
grid coordinate of GLL nodes	variable in code	common	include file (core/)
x	xm1(lx1,ly1,lz1,lelt)	/gxyz/	GEOM
y	ym1(lx1,ly1,lz1,lelt)	/gxyz/	GEOM
z	zm1(lx1,ly1,lz1,lelt)	/gxyz/	GEOM

Table 2.1: Declarations and locations for basic geometry

case	value of ldim	value of lz1
two-dimensional, $d=2$	ldim=2	lz1=1
three-dimensional, $d=3$	ldim=3	lz1=lx1=ly1$=N$

Table 2.2: Problem dimensionality and how the SIZE file can take care of so many things

Mathematical variable	variable in code	common	include file
$\mathbf{B}$	bm1(lx1,ly1,lz1,lelt)	/mass/	MASS
J	jacm1(lx1,ly1,lz1,lelt)	/giso1/	GEOM
r, s and t	zgm1(lx1,3)	/gauss/	WZ
ω_i, $i=1,\ldots,N$	wxm1(lx1)	/gauss/	WZ

Table 2.3: Declarations and locations for quadrature

(lz1=lx1=ly1=N) meshes. Every element has 2*ldim faces, and arrays in /CMTSURFLX/ (and in core commons like /GSURF/ and /FACEWZ/) store nodal vectors on faces contiguously element by element.

Subroutines that index face nodes within arrays dimensioned (lx1,ly1,lz1,lelt) storing full fields and copy such data into arrays dimensioned (lx1,lz1,2*ldim,lelt) are stored in face.f (§ 8.7). Each GLL node in each element making up $\underline{\mathbf{u}}_L$, the union of $\mathbf{u}$ for all K elements, is integrated in time in this subroutine via explicit time marching to discretizes the the left-hand-side time derivative of Equation 1.10. We choose the third-order total-variation-diminishing[1] Runge-Kutta scheme (TVDRK3) in low-storage form. We advance from t_n to $t_{n+1}=t_n+\Delta t$ via

$$\begin{aligned}
\underline{\mathbf{u}}_L^{(1)} &= \underline{\mathbf{u}}_L^{(n)} + \Delta t \mathrm{RHS}(\underline{\mathbf{u}}_L^{(n)}),\\
\underline{\mathbf{u}}_L^{(2)} &= \frac{3}{4}\underline{\mathbf{u}}_L^{(n)} + \frac{1}{4}\underline{\mathbf{u}}_L^{(1)} + \frac{1}{4}\Delta t\,\mathrm{RHS}(\underline{\mathbf{u}}_L^{(1)}),\\
\underline{\mathbf{u}}_L^{(n+1)} &= \frac{1}{3}\underline{\mathbf{u}}_L^{(n)} + \frac{2}{3}\underline{\mathbf{u}}_L^{(2)} + \frac{2}{3}\Delta t\,\mathrm{RHS}(\underline{\mathbf{u}}_L^{(2)}).
\end{aligned} \tag{2.1}$$

[1] Equivalent to the strong-stability-preserving explicit RK scheme at this order

Chapter 3

Governing equations and fluid properties

We now divide the flux function $\mathbf{H}$ into

$$\mathbf{H} = \mathbf{H}_c(\mathbf{U}) + \mathbf{H}_d(\mathbf{U}, \nabla \mathbf{U}), \tag{3.1}$$

where $\mathbf{H}_c$ is the **convective flux** function and $\mathbf{H}_d$ is the **diffusive flux** function. The convective fluxes come from the Euler equations of gas dynamics and the diffusive fluxes come from artificial viscosity used to capture shocks on spectral elements.

3.1 Euler equations of gas dynamics

We now present the Euler equations of gas dynamics In this section, bold-faced quantities are vectors in $\mathbb{R}^3$ except for the conserved variables $\mathbf{U}$,

$$\mathbf{U} = \begin{bmatrix} \phi_g \rho \\ \phi_g \rho u \\ \phi_g \rho v \\ \phi_g \rho w \\ \phi_g \rho E \end{bmatrix}, \tag{3.2}$$

which live in $\mathbb{R}^5$. Conserved variables and their inviscid fluxes[1] are weighted by the **gas volume fraction** ϕ_g. To more easily subscript flux vectors we write the m^{th}component of $\mathbf{U}$ as being governed by the conservation law

$$\frac{\partial U_m}{\partial t} + \nabla \cdot \mathbf{H}_m = R_m, \, m \in [1,5]\,. \tag{3.3}$$

The gas velocity $\mathbf{v}$ is

$$\mathbf{v} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}, \tag{3.4}$$

and ρ is the gas density, E is the mass-specific total energy $e + \frac{1}{2}|\mathbf{v}|^2$ of the gas, e is the gas internal energy, and p is the thermodynamic gas pressure.

$\mathbf{H}_m(\mathbf{U}, \nabla\mathbf{U})\mathbb{R}^{5\times 3} \rightarrow \mathbb{R}^3$ is the flux vector of equation m. Like 3.1, $\mathbf{H}_m$ is made up of a convective flux $\mathbf{H}_{m,c}(\mathbf{U})$ and a diffusive flux $\mathbf{H}_{m,d}(\mathbf{U}, \nabla\mathbf{U})$. We consider the convective fluxes first. For gas density $\mathbf{H}_{1,c}(\mathbf{U})$ is

$$\mathbf{H}_{1,c} = \phi_g \rho \mathbf{v} = [U_2, U_3, U_4]^\top\,. \tag{3.5}$$

For gas momentum U_{2-4}, the convective fluxes are

$$\mathbf{H}_{2,c} = \phi_g \begin{bmatrix} (\rho u)\, u + p \\ (\rho u)\, v \\ (\rho u)\, w \end{bmatrix}, \mathbf{H}_{3,c} = \phi_g \begin{bmatrix} (\rho v)\, u \\ (\rho v)\, v + p \\ (\rho v)\, w \end{bmatrix}, \mathbf{H}_{4,c} = \phi_g \begin{bmatrix} (\rho w)\, u \\ (\rho w)\, v \\ (\rho w)\, w + p \end{bmatrix}, \tag{3.6}$$

[1] but not their viscous fluxes

conserved variable in §3.1	index in U(ix,iy,iz,:,e)
mass $\phi_g \rho$	irg=1
x-momentum $\phi_g \rho u$	irpu=2
y-momentum $\phi_g \rho v$	irpv=3
z-momentum $\phi_g \rho w$	irpw=4
total energy per unit mass, gas $\phi_g \rho E$	irpe=5

Table 3.1: Declarations and locations for conserved variables (Equation 3.2) in CMT-nek in U in /solnconsvar/ in CMTDATA. Indices are parameters at the end of CMTDATA.

primitive variable in §3.1	variable in code	outer index	common	include file
x-velocity u, Eq. 3.4	vx(lx1,ly1,lz1,lelt)		/vptsol/	core/SOLN
y-velocity v, Eq. 3.4	vy(lx1,ly1,lz1,lelt)		/vptsol/	core/SOLN
z-velocity w, Eq. 3.4	vz(lx1,ly1,lz1,lelt)		/vptsol/	core/SOLN
thermodynamic gas pressure p	pr(lx2,ly2,lz2,lelv)		/cbm2/	core/SOLN
gas temperature T	t (lx1,ly1,lz1,lelt,ldimt)	1	/vptsol/	core/SOLN
gas density ρ	vtrans(lx1,ly1,lz1,lelt,ldimt1)	irho	/vptsol/	core/SOLN
mass-weighted specific heat ρc_p	vtrans(lx1,ly1,lz1,lelt,ldimt1)	icp	/vptsol/	core/SOLN
mass-weighted specific heat ρc_v	vtrans(lx1,ly1,lz1,lelt,ldimt1)	icv	/vptsol/	core/SOLN
isentropic sound speed a, Eq. 3.11	csound(lx1,ly1,lz1,lelt)		/cmtgasprop/	CMTDATA

Table 3.2: Declarations and locations for CMT-nek primitive variables. Outer indices are parameters in CMTDATA

and, for total energy ρE,

$$\mathbf{H}_{5,c} = \phi_g \mathbf{v} (\rho E + p) . \tag{3.7}$$

The system is closed by an equation of state,

$$[p, T] = \text{EOS} (\rho, e) . \tag{3.8}$$

Internal energy per unit mass $e = E - \frac{1}{2}|\mathbf{v}|^2$ is related to gas temperature T by the intensive property c_v, the constant-volume specific heat, such that

$$e = \int c_v(T) dT. \tag{3.9}$$

Generally, 3.9 must be solved for temperature T implicitly, iteratively, or via tabulation. For calorically perfect gases, c_v is constant. For both thermally and calorically perfect gases, pressure is obtained last via

$$p = \rho RT, \tag{3.10}$$

where the specific gas constant $R = (\gamma - 1) c_v$ requires the specification of $\gamma = c_p/c_v$, the ratio of constant-pressure specific heat c_p to c_v. Finally, the sound speed is

$$a = \sqrt{\frac{\gamma p}{\rho}}. \tag{3.11}$$

Vectors of nodal values are usually stored consecutively in multidimensional arrays whose "outermost" index is fixed for that particular quantity. As an exception, conserved variables are dimensioned element-outermost, and the second-to-last dimension is fixed for a given conserved variable. The indexing for this purpose is given in Table 3.1.

Table 3.2 lays out where some of the physical variables are declared and stored. We recycle many of nek5000's arrays, with some abuse. In the classic $P_N P_{N-2}$ spectral element method, pressure is stored at nodal values on a separate mesh (m2="mesh 2") of Gauss-Legendre(GL) and not GLL quadrature nodes. CMT-nek may need this mesh to follow Fisher & Carpenter in a DGSEM formulation with the summation-by-parts property. For now, however, pressure is not staggered. CMT-nek must be run in the $P_N P_N$ mode with lx2=lx1, and using mesh-1 metrics for everything, including pr, which is not correct for nek5000 in general.

Fluxes in the Euler equations (Equation 3.5 through 3.7) are functions of **primitive variables** (Table 3.2) in addition to the primary unknowns, the conserved variables. (Equation 3.2).

Chapter 4

Right-hand-side evaluation: volume terms

4.1 Two-point flux functions

Gaussian quadrature on N GLL points exactly integrates a polynomial of order $2(N-1)-1$. However, nonlinear flux functions like 3.6 produce integrands that are rational functions of $\mathbf{U}$ since we are dividing by density to get velocity from momentum. Gaussian quadrature at any order is not exact for rational integrands. Worse arithmetic than mere division is required by most state equations. integrated on only N points. These errors tend to accumulate with time, and something must be done to stabilize the scheme against their deleterious effects.

The major motivation for §1.2 is a robust stabilization scheme for doing quadrature on these fluxes. Equation 1.16 has been proven to guarantee that if the two-point form $\mathbf{F}^{\#}$ conserves kinetic energy, then a high-order SBP operator using it in Equations 1.19 through 1.21 will do so exactly and discretely. More importantly, if $\mathbf{F}^{\#}$ is **entropy-stable** (provably increases physical entropy or conserves it), then high-order SBP operators using it in DGSEM will be discretely entropy-stable as well. A growing body of tests **?** has shown that these discrete stability properties are recovered without loss of formal order of accuracy in constructed solutions and turbulent flows. Furthermore, these properties translate to better stability at lower cost than in traditional "overintegration**?**" or "dealiasing" techniques.

We finally give some examples of the two-point flux functions that DGSEM depends on. As usual, details and motivation may be found in Gassner, Winters and Kopriva.

4.1.1 Kennedy and Gruber

The flux in Equation (3.10) of Gassner, Winters & Kopriva is the kinetic-energy-preserving skew-symmetric split form of Kennedy & Gruber**?** reformulated for SBP operators in the form of Equation 1.16. In each coordinate direction this flux is, for all 5 conserved variables,

$$\mathbf{F}_1^{\#}\left(\mathbf{u}_{(ijk)},\mathbf{u}_{(ljk)}\right),\mathbf{F}_2^{\#}\left(\mathbf{u}_{(ijk)},\mathbf{u}_{(ilk)}\right),\mathbf{F}_3^{\#}\left(\mathbf{u}_{(ijk)},\mathbf{u}_{(ijl)}\right), \tag{4.1}$$

$$\mathbf{F}_1^{\#}=\begin{bmatrix}\hat{\rho}\hat{u}\\ \hat{\rho}\hat{u}^2+\hat{p}\\ \hat{\rho}\hat{u}\hat{v}\\ \hat{\rho}\hat{u}\hat{w}\\ \hat{u}\left(\hat{\rho}\hat{e}+\hat{p}\right)\end{bmatrix},\mathbf{F}_2^{\#}=\begin{bmatrix}\hat{\rho}\hat{v}\\ \hat{\rho}\hat{v}\hat{u}\\ \hat{\rho}\hat{v}^2+\hat{p}\\ \hat{\rho}\hat{v}\hat{w}\\ \hat{v}\left(\hat{\rho}\hat{e}+\hat{p}\right)\end{bmatrix},\mathbf{F}_3^{\#}=\begin{bmatrix}\hat{\rho}\hat{w}\\ \hat{\rho}\hat{w}\hat{u}\\ \hat{\rho}\hat{w}\hat{v}\\ \hat{\rho}\hat{w}^2+\hat{p}\\ \hat{w}\left(\hat{\rho}\hat{e}+\hat{p}\right)\end{bmatrix}, \tag{4.2}$$

where the "hat" variables are actually function of the indicated quantity at the two points upon which $\mathbf{F}_k^{\#}$ acts. For the Kennedy-Gruber and other energy-stable fluxes,

$$\hat{f}\left(\mathbf{f}_{(ijk)},\mathbf{f}_{(ljk)}\right)\equiv\{\{f\}\}_{((i,l)jk)}, \tag{4.3}$$

where Equation 1.22 defines the "$\{\{\}\}$" averaging operator. Similar definitions exist for the grid lines in other directions on the reference element. Thus, the Kennedy and Gruber flux is a **product of averages** of quantities at the two points used to evaluate $\mathbf{F}^{\#}$ in Equation 1.16.

The volume integral term for the diffusive fluxes $\mathsf{K}_{m,d}$ is, for the m^{th}conserved variable,

$$\mathsf{K}_{m,d} \equiv \mathbf{B}^{-1}\left(\mathbf{D}_{r_l}^{\top}\left[\text{diag}\left(\frac{\partial r_l}{\partial x_i}\right)\mathbf{B}\left(\mathscr{A}_{mijL}\left[\text{diag}\left(\frac{\partial r_k}{\partial x_j}\right)\mathbf{D}_{r_k}\mathbf{u}_L\right]\right)\right]\right), \tag{4.4}$$

where Einstein summation convention is used on all Roman subscripts. The ranges of these subscripts follow their ordering in $\mathscr{A}_{mijL}$ (**??**), and $\mathbf{u}_L$ refers to a nodal vector (1.6) of the L^{th}conserved variable in 3.2.

Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 7

Module Documentation

7.1 Volume integral for inviscid fluxes

Functions/Subroutines

- subroutine convective_cmt (e)

  Evaluates inviscid volume terms for all toteq equations in two-point split form (Equation∼shortvol}and adds them to res1(:,:,:,e,:).
- subroutine fluxdiv_2point_noscr (res, fcons, e, ja)

  Evaluates the two-point split form (Equations∼splitr} through splitt}) of the volume integral $\int v\nabla\cdot\mathbf{H}^c dV$ and the discontinuous surface flux $\oint v\mathbf{H}^c\cdot\mathbf{n}dA$ for the inviscid flux function in a single element.
- subroutine fluxdiv_strong_contra (e)

  Computes $\int v\nabla\cdot\mathbf{H}^c dV$ in aliased strong form for element e, and increments res1 with it.
- subroutine evaluate_aliased_conv_h (e)

  Evaluates consistent (i.e. $F^{\#}(U_{i,j,k},U_{l,j,k})=H_{(eq),i},i=l$) flux function at all GLL nodes and stores it in convh.
- subroutine fluxdiv_dealiased_weak_chain (e)

  $(\nabla v)\cdot\mathbf{H}^c=\mathscr{I}^\mathsf{T}\mathbf{D}^\mathsf{T}\cdots$ *for equation eq, element e*
- subroutine contravariant_flux (frst, fxyz, ja, nel)

  Transforms consistent (i.e. $F^{\#}(U_{i,j,k},U_{l,j,k})=H_{(eq),i},i=l$) flux for one conserved variable to the contravariant frame by contracting H_i with J diag $(\partial r_j/\partial x_i)$.

7.1.1 Detailed Description

7.1.2 Function/Subroutine Documentation

7.1.2.1 subroutine contravariant_flux (real, dimension(nx1∗ny1∗nz1,ldim,nel) *frst,* real, dimension(nx1∗ny1∗nz1,ldim,nel) *fxyz,* real, dimension(nx1∗ny1∗nz1,ldim∗ldim,nel) *ja,* integer *nel*)

Transforms consistent (i.e. $F^{\#}(U_{i,j,k},U_{l,j,k})=H_{(eq),i},i=l$) flux for one conserved variable to the contravariant frame by contracting H_i with J diag $(\partial r_j/\partial x_i)$.

Parameters

ja	Metrics for the nel elements (intent(in))
frst	Spatial vector of contravariant flux for one variable, nel elements (intent(out))
fxyz	Spatial vector of physical flux for one variable, nel elements (intent(in))

nel	Number of elements (intent(in))

7.1.2.2 subroutine convective_cmt (integer *e*)

Evaluates inviscid volume terms for all toteq equations in two-point split form (Equation∼shortvol}and adds them to res1(:,:,:,e,:).

Parameters

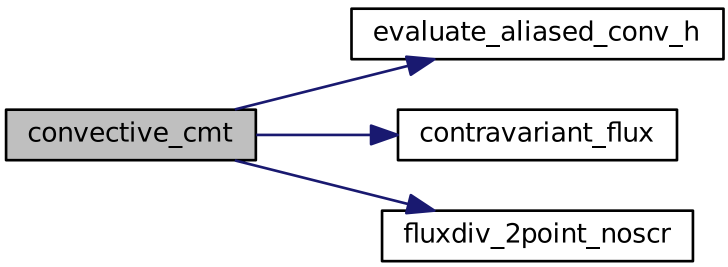
e	element e

Consistent flux (i.e. $F^{\#}(U_{i,j,k}, U_{l,j,k}), i = l$) is computed by evaluate_aliased_conv_h and stored in convh for a single element.

Transform the consistent flux to the contravariant frame in the reference element.

Compute the flux divergence in 2-point form using the flux function defined in cmt_usrflx and the consistent contravariant flux stored in convh. store it in totalh and add it to res1(:,:,:,e,:)

Here is the call graph for this function:



7.1.2.3 subroutine evaluate_aliased_conv_h (integer *e*)

Evaluates consistent (i.e. $F^{\#}(U_{i,j,k}, U_{l,j,k}) = H_{(eq),i}, i = l$) flux function at all GLL nodes and stores it in convh.

Parameters

e	integer index of the element (intent(in))

7.1.2.4 subroutine fluxdiv_2point_noscr (real, dimension(lx1,ly1,lz1,lelt,toteq) *res,* real, dimension(lx1,ly1,lz1,3,toteq) *fcons,* integer *e,* real, dimension(lx1,ly1,lz1,ldim,ldim) *ja*)

Evaluates the two-point split form (Equations∼splitr} through splitt}) of the volume integral $\int v\nabla\cdot\mathbf{H}^c dV$ and the discontinuous surface flux $\oint v\mathbf{H}^c\cdot\mathbf{n}dA$ for the inviscid flux function in a single element.

Parameters

res	RHS for all equations, all elements (intent(inout))
ja	Metrics for the e'th elements (intent(in))
fcons	Consistent contravariant flux $Jdiag\left(\partial r_i/\partial x_k\right)\mathbf{H}_k$ (intent(in))

Both $\int v\nabla\cdot\mathbf{H}^c dV$ and flux $\oint v\mathbf{H}^c\cdot\mathbf{n}dA$ for the inviscid flux are added to res1 for the element e. The volume integrand is approximated by $\sum D_{il}F^{\#}(z_{i,j,k},z_{l,j,k})$ in each of the ndim directions of the reference element. The two-point flux function $F^{\#}(z_1,z_2)$ is taken from fluxfn.f and specified in the usr file in cmt_usr2pt.

The parameter vector z is computed from primitive variables in element e of SOLN and stored one element at a time in /scrns/ zaux according to cmt_usrz. U is transposed and stored for element e in ut.

7.1.2.5 subroutine fluxdiv_strong_contra (integer *e*)

Computes $\int v\nabla\cdot\mathbf{H}^c dV$ in aliased strong form for element e, and increments res1 with it.

Parameters

e	integer index of the element (intent(in))

7.2 Surface integrals due to boundary conditions

Functions/Subroutines

- subroutine inviscidbc (flux)

  Determining rind state for Dirichlet boundary conditions.
- subroutine bcmask_cmt (bmsk)

  Mask to make sure Fsharp doesn't clobber boundary faces, where gs_op is null This routine intents to take a real array for all face points, bmask, and only zero out faces on boundaries. It is thus not limited to an array only of indicators.
- subroutine bcflux (flux, agradu, qminus)

  Determining IGU contribution to boundary flux. 0 for artificial viscosity, and strictly interior for physical viscosity.
- subroutine a5adiabatic_wall (eflx, f, e, dU, wstate)

  computes boundary flux for adiabatic wall in igu
- subroutine a51duadia (flux, f, ie, dU, wstate)

  same as A51 for volume flux (x-direction viscous flux of energy, but
- subroutine a52duadia (flux, f, ie, dU, wstate)

  same as A51 for volume flux (y-direction viscous flux of energy, but
- subroutine a53duadia (flux, f, ie, dU, wstate)

  same as A51 for volume flux (z-direction viscous flux of energy, but
- subroutine imqqtu_dirichlet (umubc, wminus, wplus)

  $\left(\mathbf{I}-1/2QQ^T\right)\mathbf{U}\rightarrow\mathbf{U}^- - \mathbf{U}^D$ *on Dirichlet boundaries. Currently only modifies* $\mathbf{U}^+$ *in the case of walls.*

7.2.1 Detailed Description

7.2.2 Function/Subroutine Documentation

7.2.2.1 subroutine a51duadia (real, dimension (lx1∗ly1∗lz1) *flux,* integer *f,* *ie,* real, dimension (lx1∗lz1,2∗ldim,nelt,toteq,3) *dU,* real, dimension(lx1∗lz1,2∗ldim,nelt,nqq) *wstate*)

same as A51 for volume flux (x-direction viscous flux of energy, but

1. uses wstate for contiguous storage of data on faces.
2. locally sets K=0 to prevent heat transfer through adiabatic walls

7.2.2.2 subroutine a52duadia (real, dimension (lx1∗ly1∗lz1) *flux,* integer *f,* *ie,* real, dimension (lx1∗lz1,2∗ldim,nelt,toteq,3) *dU,* real, dimension(lx1∗lz1,2∗ldim,nelt,nqq) *wstate*)

same as A51 for volume flux (y-direction viscous flux of energy, but

1. uses wstate for contiguous storage of data on faces.
2. locally sets K=0 to prevent heat transfer through adiabatic walls

7.2.2.3 subroutine a53duadia (real, dimension (lx1∗ly1∗lz1) *flux,* integer *f,* *ie,* real, dimension (lx1∗lz1,2∗ldim,nelt,toteq,3) *dU,* real, dimension(lx1∗lz1,2∗ldim,nelt,nqq) *wstate*)

same as A51 for volume flux (z-direction viscous flux of energy, but

1. uses wstate for contiguous storage of data on faces.
2. locally sets K=0 to prevent heat transfer through adiabatic walls

7.2.2.4 subroutine imqqtu_dirichlet (real, dimension (lx1*lz1,2*ldim,nelt,toteq) *umubc,* real, dimension(lx1*lz1,2*ldim,nelt,nqq) *wminus,* real, dimension (lx1*lz1,2*ldim,nelt,nqq) *wplus*)

$\left(\mathbf{I}-1/2QQ^T\right)\mathbf{U} \rightarrow \mathbf{U}^{-}-\mathbf{U}^{D}$ on Dirichlet boundaries. Currently only modifies $\mathbf{U}^{+}$ in the case of walls.

Parameters

umubc	$\mathbf{U}^{-} - \mathbf{U}^{D}$ (intent(out))
wminus	interior values of primitive variables on boundary faces (intent(in))
wplus	exterior values of primitive variables on faces with Dirichlet BC. Filled with desired boundary condition values (intent(inout)) on entry, but modified for adiabatic walls.

7.3 Volume integral for viscous fluxes

Functions/Subroutines

- subroutine half_iku_cmt (res, diffh, e)

  Compute the integrand $\mathbf{D}^T\mathbf{H}^d$ *of the weak-form volume integral and store it in res1 one element per call.*

- subroutine viscous_cmt (e, eq)

  Volume integral for diffusive terms.

7.3.1 Detailed Description

7.3.2 Function/Subroutine Documentation

7.3.2.1 subroutine half_iku_cmt (real, dimension(lx1,ly1,lz1) *res*, real, dimension(lx1∗ly1∗lz1,ldim) *diffh*, integer *e*)

Compute the integrand $\mathbf{D}^T\mathbf{H}^d$ of the weak-form volume integral and store it in res1 one element per call.

Parameters

e	index of element under consideration (intent(in))
res	res+= $\mathbf{D}^T\mathbf{H}^{(d)}$. Actual argument is a single element of res1 for a single equation (intent(inout))
diffh	viscous flux $\mathbf{H}^{(d)}$ for a single element and single equation. Overwritten (intent(inout))

$\mathbf{MH}^d$ in diffh

$\mathbf{D}^T\mathbf{MH}^d$ in rscr for element e.

$\mathbf{M}^{-1}\mathbf{D}^T\mathbf{MH}^d$ in rscr for element e.

add rscr to res

7.3.2.2 subroutine viscous_cmt (integer *e*, integer *eq*)

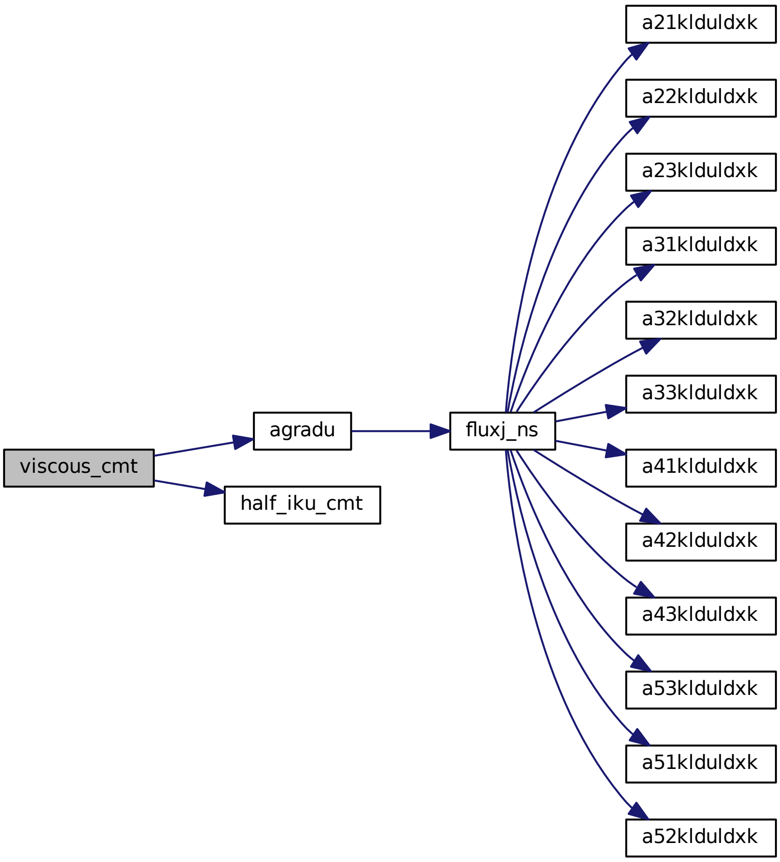
Volume integral for diffusive terms.

Compute $\mathbf{H}^d = A\nabla U$ and store it in diffh for element e.

Store faces of $\mathbf{H}^d$ in /CMTSURFLX/ for $\mathbf{I}_{GU}$ or BR1.

Compute the integrand $\mathbf{D}^T\mathbf{H}^d$ of the weak-form volume integral and store it in res1.

Here is the call graph for this function:



7.4 Jacobians for viscous fluxes

Functions/Subroutines

- subroutine agradu (flux, du, e, eq)

    Transforms the gradients of conserved variables $\nabla \mathbf{U}$ to the viscous flux $\mathbf{H}^{(d)}$ in a single element for a single equation. Particular choice of viscous stress tensor is currently hardcoded for Navier-Stokes.
- subroutine fluxj_ns (flux, gradu, e, eq)

    $\tau_{ij} = 2\mu\sigma_{ij} + \lambda\Delta\delta_{ij}$ *Navier-Stokes. uservp provides properties stored in SOLN. Implemented via maxima-generated code.*
- subroutine compute_transport_props

    Fill vdiff with transport properties. Hardcoded indices for Navier-Stokes Used for both artificial and physical viscosities.
- subroutine a51kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.
- subroutine a52kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.
- subroutine a53kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.
- subroutine a21kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.
- subroutine a22kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.
- subroutine a23kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.
- subroutine a31kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.
- subroutine a32kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.
- subroutine a33kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.
- subroutine a41kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.
- subroutine a42kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.
- subroutine a43kldUldxk (flux, dU, ie)

    WRITE OUT NOTATION.

7.4.1 Detailed Description

7.4.2 Function/Subroutine Documentation

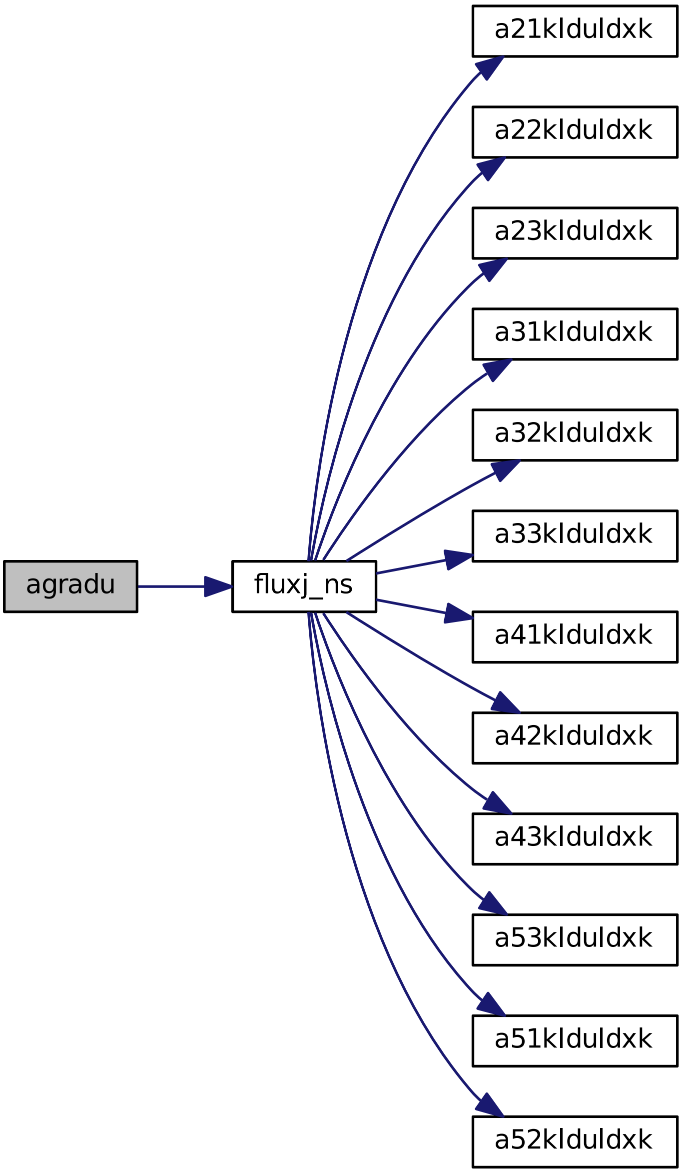
7.4.2.1 subroutine agradu (real, dimension(lx1∗ly1∗lz1,ldim) *flux*, real, dimension(lx1∗ly1∗lz1,3,toteq) *du*, integer *e*, integer *eq*)

Transforms the gradients of conserved variables $\nabla \mathbf{U}$ to the viscous flux $\mathbf{H}^{(d)}$ in a single element for a single equation. Particular choice of viscous stress tensor is currently hardcoded for Navier-Stokes.

Parameters

e	index of element for primitive variables within different flux jacobians (intent(in))
eq	index of conserved variable whose viscous flux is being computed. (intent(in))
du	gradient of conserved variables $\partial U_i/\partial x_j$ (intent(in))
flux	flux = $\mathbf{H}^{(d)} = \mathscr{A}\nabla\mathbf{U}$ (intent(out))

Here is the call graph for this function:



7.4.2.2 subroutine compute_transport_props ( )

Fill vdiff with transport properties. Hardcoded indices for Navier-Stokes Used for both artificial and physical viscosities.

Indexes the element inside SOLN (intent(in))

Here is the call graph for this function:

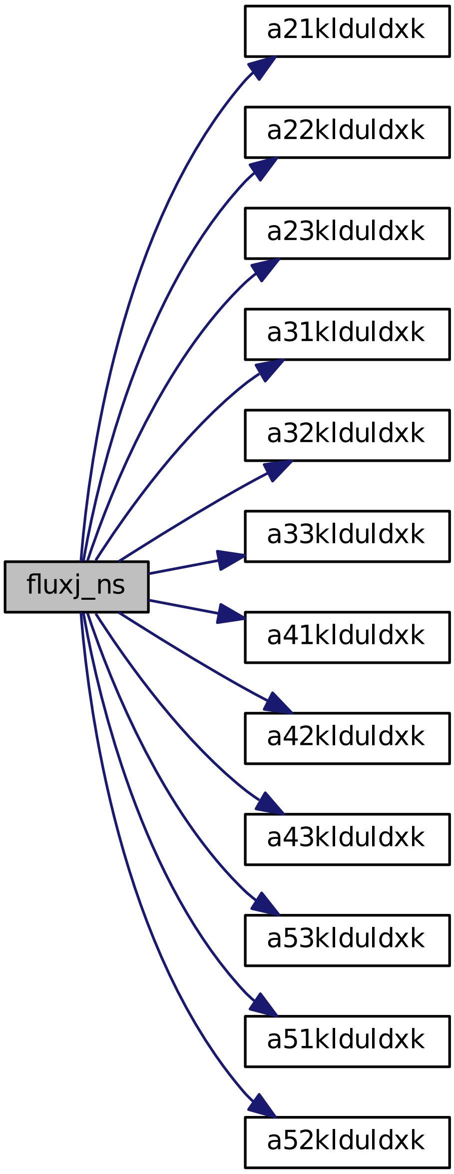
7.4.2.3 subroutine fluxj_ns (real, dimension(lx1*ly1*lz1,ldim) *flux*, real, dimension(lx1*ly1*lz1,3,toteq) *gradu*, integer *e*, integer *eq*)

$\tau_{ij} = 2\mu\sigma_{ij} + \lambda\Delta\delta_{ij}$ Navier-Stokes. uservp provides properties stored in SOLN. Implemented via maxima-generated code.

Parameters

e	index of element under consideration (intent(in))
eq	index of conserved variable whose viscous flux is being computed. (intent(in))
flux	flux = $\mathbf{H}^{(d)} = \mathscr{A}\nabla\mathbf{U}$ (intent(out))
gradu	gradient of conserved variables $\nabla\mathbf{U}$ (intent(in))

Here is the call graph for this function:



7.5 Inviscid surface terms

Functions/Subroutines

- subroutine ausm_fluxfunction (ntot, nx, ny, nz, nm, fs, rl, ul, vl, wl, pl, al, tl, rr, ur, vr, wr, pr, ar, tr, flx, el, er)

  Computes inviscid numerical surface flux from AUSM+ Riemann solver.
- subroutine inflow_df (f, e, wm, wp, um, up, nvar)

  more conventional Dolejsi & Feistauer (2015) Section 8.3.2.2 "physical" boundary conditions. Also encountered in Hartmann & Houston (2006). A poor default.
- subroutine outflow_df (f, e, wm, wp, um, up, nvar)

  Dolejsi & Feistauer (2015) Section 8.3.2.2. Very rudimentary "physical" boundary conditions. Also encountered in Hartmann & Houston (2006). A poor default.
- subroutine fillujumpu

  overwrite beginning of /CMTSURFLX/ with $-[[\mathbf{U}]]$ for flux of auxiliary variable in the viscous flux of Bassi and Rebay computed in br1auxflux.
- subroutine fluxes_full_field_kg

  Restrict and copy face data and compute inviscid numerical flux $\oint \mathbf{H}^{c*} \cdot \mathbf{n} dA$ on face points. This particular wrapper is for the symmetric flux of Kennedy and Gruber.
- subroutine faceu (ivar, yourface)

  Restrict and copy face data for conserved variables U_{ivar}. Wraps full2face_cmt for each element; a single call to full2face with U will not work because element varies with the outermost index of the u array.
- subroutine fillq (jvar, field, wminus, yourface)

  Restrict and copy face data for one full field and store it in index jvar in wminus.
- subroutine dg_face_avg (mine, nf, nstate, handle)

  Overwrite values stored at points on faces with the average with its values in the neighboring face without duplication. Replaces $mine$ with $\{\{mine\}\}$.
- subroutine face_state_commo (mine, yours, nf, nstate, handle)

  Sends face values v^- stored in "mine" to the neighbor that shares that face and copies the neighbor's values v^+ at each face into "yours.".
- subroutine avg_and_jump (avg, jump, scratch, nf, nstate, handle)

  Overwrites w^- at interior face points stored in avg with $\{\{w\}\}$. jump gets filled with $[[w]]$.

7.5.1 Detailed Description

7.5.2 Function/Subroutine Documentation

7.5.2.1 subroutine avg_and_jump (real, dimension($*$) *avg*, real, dimension($*$) *jump*, real, dimension($*$) *scratch*, integer *nf*, integer *nstate*, integer *handle*)

Overwrites w^- at interior face points stored in avg with $\{\{w\}\}$. jump gets filled with $[[w]]$.

Parameters

handle	integer handle for gs_op. needs to be set by fgslib_gs_setup call in setup_cmt_gs call (intent(in))
nf	Total number of face points on all faces in the domain. (intent(in), but should always be lx1∗lz1∗2∗ldim∗nelt)
nstate	Number of distinct fields whose copies are to be transfered between neighboring elements (intent(in))
avg	Real buffer (intent(inout)) w^- on input, $\{\{w\}\}$ on output

jump	Real buffer (intent(out)) w^- on input, $[[w]]$ on output
scratch	Real scratch (intent(out))

7.5.2.2 subroutine dg_face_avg (real, dimension(∗) *mine,* integer *nf,* integer *nstate,* integer *handle*)

Overwrite values stored at points on faces with the average with its values in the neighboring face without duplication. Replaces $mine$ with $\{\{mine\}\}$.

Parameters

handle	integer handle for gs_op. needs to be set by fgslib_gs_setup call in setup_cmt_gs call (intent(in))
nf	Total number of face points on all faces in the domain. (intent(in), but should always be lx1∗lz1∗2∗ldim∗nelt)
nstate	Number of distinct fields whose averages between neighboring elements are to be computed and stored at face points (intent(in))
mine	Buffer, but should be some large array within /CMTSURFLX/ (real, intent(inout))

7.5.2.3 subroutine face_state_commo (real, dimension(∗) *mine,* real, dimension(∗) *yours,* integer *nf,* integer *nstate,* integer *handle*)

Sends face values v^- stored in “mine” to the neighbor that shares that face and copies the neighbor’s values v^+ at each face into “yours.”.

Parameters

handle	integer handle for gs_op. needs to be set by fgslib_gs_setup call in setup_cmt_gs call (intent(in))
nf	Total number of face points on all faces in the domain. (intent(in), but should always be lx1∗lz1∗2∗ldim∗nelt)
nstate	Number of distinct fields whose copies are to be transfered between neighboring elements (intent(in))
mine	Buffer storing interior states v^- in /CMTSURFLX/ (real, intent(in))
yours	Buffer storing exterior/neighbor states v^+ in /CMTSURFLX/ (real, intent(out))

7.5.2.4 subroutine faceu (integer *ivar,* real, dimension(lx1,lz1,2∗ldim,nelt) *yourface*)

Restrict and copy face data for conserved variables U_{ivar}. Wraps full2face_cmt for each element; a single call to full2face with U will not work because element varies with the outermost index of the u array.

Parameters

ivar	index of variable within U. ivar=1 for $\phi\rho$, ivar=2-4 for $\phi\rho u_i$, etc.
yourface	contiguous pile of faces for the $ivar^{th}$ conserved variable. (dimension(lx1,lz1,2∗ldim,nelt), intent(out))

7.5.2.5 subroutine fillq (integer *jvar,* real, dimension(lx1,ly1,lz1,nelt) *field,* real, dimension(lx1∗lz1∗2∗ldim∗nelt,∗) *wminus,* real, dimension(lx1∗lz1∗2∗ldim∗nelt) *yourface*)

Restrict and copy face data for one full field and store it in index jvar in wminus.

Parameters

jvar	index within wminus where field array at points lying on faces will be stored (intent(in))
field	full array of values at each GLL point in all elements (intent(in), dimension(lx1,ly1,lz1,nelt))
wminus	contiguous storage for multiple fields at all GLL nodes lying on faces of each element (intent(out), dimension(lx1∗lz1∗2∗ldim∗nelt,∗))
yourface	contiguous scratch array for a single field at all GLL nodes lying on faces of each element (intent(out), dimension(lx1∗lz1∗2∗ldim∗nelt))

7.5.2.6 subroutine fluxes_full_field_kg ( )

Restrict and copy face data and compute inviscid numerical flux $\oint \mathbf{H}^{c*} \cdot \mathbf{n} dA$ on face points. This particular wrapper is for the symmetric flux of Kennedy and Gruber.

"heresize" and "hdsize" come from a failed attempt at managing memory in CMTSURFLX by redeclaration that was abandoned before the two-point split form. They need to be taken care of in CMTSIZE and consistent with the desired subroutine

duplicate one conserved variable (from neighboring elements) at a time for jumps in LLF

store it before phi, which lives at jph=nqq

Stabilization first. Local Lax Friedrichs for equations 1 through 4 (mass & momentum)

ONLY needed by Kennedy-Gruber (2008) as written. This is done in fstab for Chandrashekar (2013), but overwrites jsnd for KG and friends.

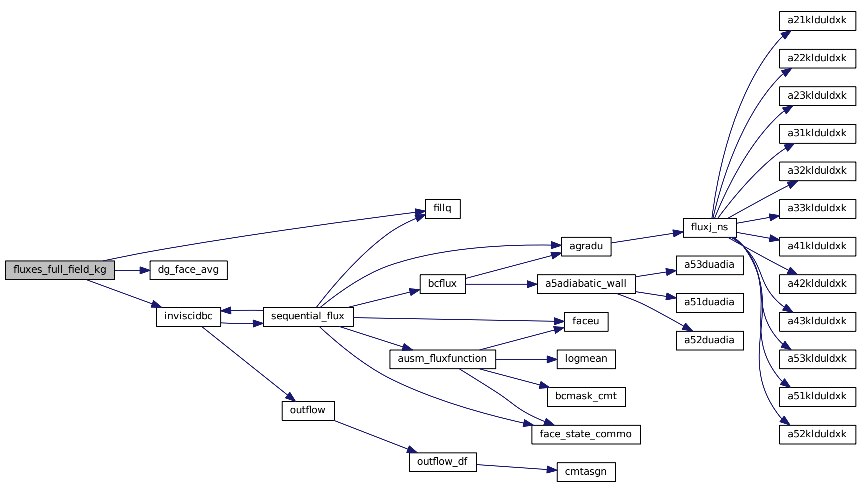
$\mathbf{q}^- \rightarrow \mathbf{z}^-$. Kennedy-Gruber, Pirozzoli, and most energy- conserving fluxes have $\mathbf{z} = \mathbf{q}$, so I just divide total energy by U_1 here since Kennedy-Gruber needs E.

$\mathbf{z}^- \rightarrow \hat{z}^-$, which is $\{\{\mathbf{z}\}\}$ for Kennedy-Gruber, Pirozzoli, and some parts of other energy-conserving fluxes.

$\hat{z} \rightarrow F^{\#}$. Some parameter-vector stuff can go here too as long as it's all local to a given element.

Now do all fluxes for all boundaries, both $F^{\#}$ and stabilized

Here is the call graph for this function:



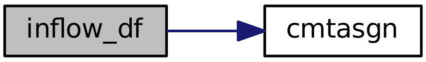
7.5.2.7 subroutine inflow_df (integer *f*, integer *e*, real, dimension(nvar,lx1∗lz1) *wm*, real, dimension(nvar,lx1∗lz1) *wp*, real, dimension(toteq,lx1∗lz1) *um*, real, dimension(toteq,lx1∗lz1) *up*, integer *nvar*)

more conventional Dolejsi & Feistauer (2015) Section 8.3.2.2 “physical” boundary conditions. Also encountered in Hartmann & Houston (2006). A poor default.

Parameters

f	face index from 1 to 2∗ldim
e	element index
nvar	number of primitive variables
wm	primitive variables from flow solution (dimension(nvar,lx1∗lz1),intent(in))
wp	external dirichlet state's primitive variables (dimension(nvar,lx1∗lz1),intent(out))
um	conserved variables from flow solution (dimension(toteq,lx1∗lz1),intent(in))
up	external dirichlet state's conserved variables (dimension(toteq,lx1∗lz1),intent(out))

Here is the call graph for this function:



7.5.2.8 subroutine outflow_df (integer *f*, integer *e*, real, dimension(nvar,lx1∗lz1) *wm*, real, dimension(nvar,lx1∗lz1) *wp*, real, dimension(toteq,lx1∗lz1) *um*, real, dimension(toteq,lx1∗lz1) *up*, integer *nvar*)

Dolejsi & Feistauer (2015) Section 8.3.2.2. Very rudimentary "physical" boundary conditions. Also encountered in Hartmann & Houston (2006). A poor default.

Parameters

f	face index from 1 to 2∗ldim
e	element index
nvar	number of primitive variables
wm	primitive variables from flow solution (dimension(nvar,lx1∗lz1),intent(in))
wp	external dirichlet state's primitive variables (dimension(nvar,lx1∗lz1),intent(out))
um	conserved variables from flow solution (dimension(toteq,lx1∗lz1),intent(in))
up	external dirichlet state's conserved variables (dimension(toteq,lx1∗lz1),intent(out))

Here is the call graph for this function:

7.6 Viscous surface terms

Functions/Subroutines

- subroutine br1auxflux (e, flux, ujump)

  add BR1 auxiliary flux $\frac{1}{2}\left(\mathbf{U}^{+}-\mathbf{U}^{-}\right)$ *to the gradient for a single element*

- subroutine imqqtu (ummcu, uminus, uplus)

  ummcu = $\mathbf{U}^{-}-\{\{\mathbf{U}\}\}$

- subroutine igtu_cmt (qminus, ummcu, hface)

  Computes $G^{T}U$*, the volume integral of* $[[u]]\cdot\{\{\nabla v\}\}$*, and increments res1 with it.*

7.6.1 Detailed Description

7.6.2 Function/Subroutine Documentation

7.6.2.1 subroutine br1auxflux (integer *e*, real, dimension(lx1∗ly1∗lz1,ldim) *flux*, real, dimension(lx1∗lz1∗2∗ldim,nelt) *ujump*)

add BR1 auxiliary flux $\frac{1}{2}\left(\mathbf{U}^{+}-\mathbf{U}^{-}\right)$ to the gradient for a single element

Parameters

e	Indexes the element inside ujump, GEOM and DG (intent(in))
ujump	Jump in a single conserved variable $[[U]]$ on all faces of all elements (intent(in))
flux	On entry: gradient $\partial U_i^{-}/\partial x_k$ of a single conserved variable. On exit: all three components of auxiliary variable $\mathbf{S}$ of a single conserved variable. (intent(inout))

7.6.2.2 subroutine igtu_cmt (real, dimension(lx1∗lz1,2∗ldim,nelt,∗) *qminus*, real, dimension(lx1∗lz1∗2∗ldim,nelt,toteq) *ummcu*, real, dimension(lx1∗lz1∗2∗ldim∗nelt,toteq,3) *hface*)

Computes $G^{T}U$, the volume integral of $[[u]]\cdot\{\{\nabla v\}\}$, and increments res1 with it.

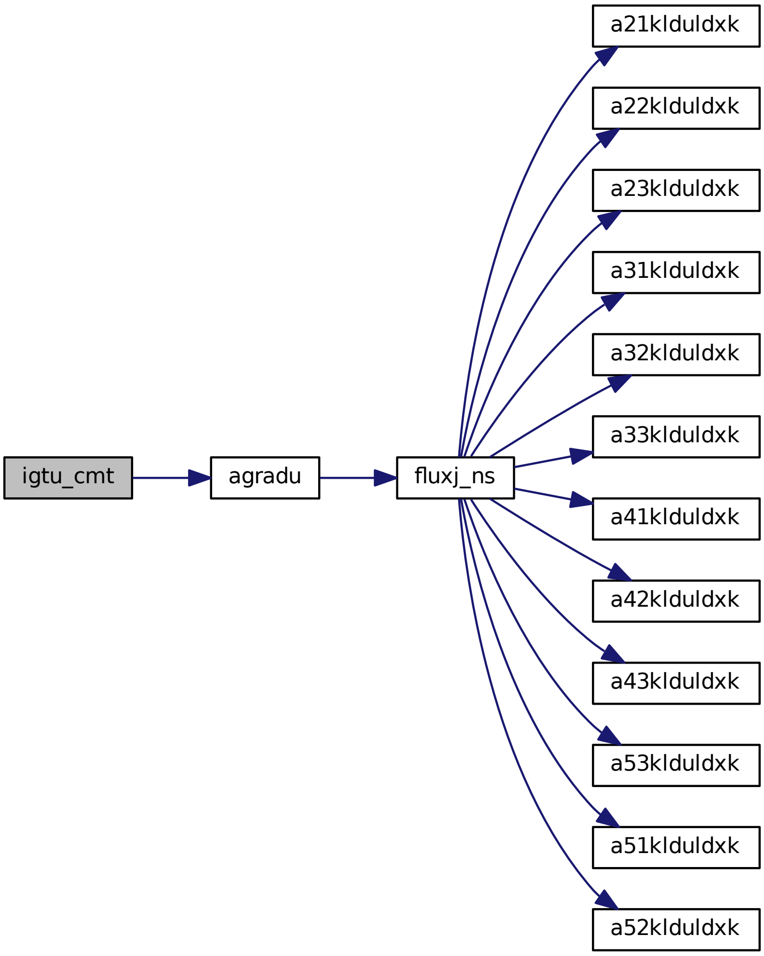
Parameters

qminus	qminus contains values of state variables indexed by j∗ at each element's own face points ummcu contains $U-\{\{U\}\}$

Symmetric interior penalty method (SIP) and Baumann-Oden have opposite signs for this term. The sign is HARD-CODED here.

gradm1_t uses /ctmp1/

Here is the call graph for this function:



7.6.2.3 subroutine imqqtu (real, dimension (lx1∗lz1∗2∗ldim∗nelt,toteq) *ummcu*, real, dimension(lx1∗lz1∗2∗ldim∗nelt,toteq) *uminus*, real, dimension (lx1∗lz1∗2∗ldim∗nelt,toteq) *uplus*)

ummcu = $\mathbf{U}^- - \{\{\mathbf{U}\}\}$

Parameters

ummcu	ummcu = $\mathbf{U}^- - \{\{\mathbf{U}\}\}$ for all faces (intent(out))
uminus	$\mathbf{U}^-$ for all faces on all elements (intent(in))
uplus	Neighbor values $\mathbf{U}^+$ for all faces on all elements (intent(in))

7.7 Flux functions and wrappers

- subroutine sequential_flux (flux, wminus, wplus, uminus, uplus, jaminus, japlus, fluxfunction, nstate, npt)

  Calls two-point external fluxfunction $F^{\#}(U^{-},U^{+})$ at npt points Mostly intended to allow quantity-innermost volume flux functions to be used where needed for surface fluxes at boundary points, after $*$bc routines provide Dirichlet “rind” states in wplus and uplus.

- subroutine **inviscidflux** (wminus, wplus, flux, nstate, nflux)
- subroutine **surface_integral_full** (vol, flux)
- subroutine **diffh2graduf** (e, eq, graduf)
- subroutine **diffh2face** (e, eq, diffhf)
- subroutine **igu_cmt** (flxscr, gdudxk, wminus)
- subroutine **igu_dirichlet** (flux, agradu)
- subroutine **br1primary** (flux, gdudxk)
- subroutine **agradu_normal_flux** (flux, graduf)
- subroutine **br1bc** (flux)
- subroutine **bcflux_br1** (flux, f, e)
- subroutine **strong_sfc_flux** (flux, vflx, e, eq)
- subroutine **fluxes_full_field_chold**
- subroutine **fluxes_full_field_old**
- subroutine **inviscidfluxrot** (wminus, wplus, flux, nstate, nflux)
- subroutine gtu_wrapper (fatface)

7.7.1 Detailed Description

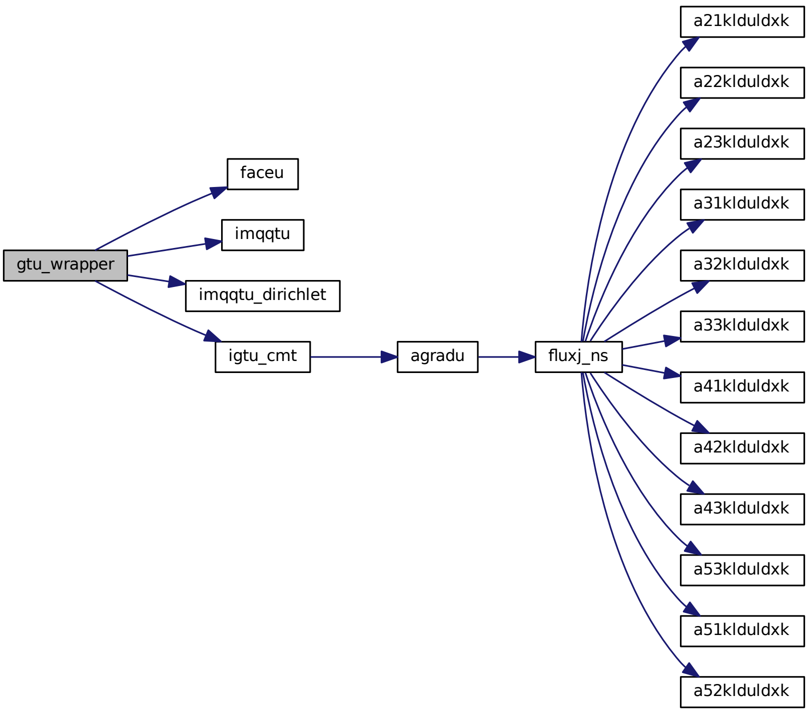
7.7.2 Function/Subroutine Documentation

7.7.2.1 subroutine gtu_wrapper (real, dimension($*$) *fatface*)

res1+= $\int_{\Gamma}\{\{\mathbf{A}^{\intercal}\nabla v\}\}\cdot[\mathbf{U}]\,dA$

res1+= $\int(\nabla v)\cdot\left(\mathbf{H}^{c}+\mathbf{H}^{d}\right)dV$ for each equation (inner), one element at a time (outer)

Here is the call graph for this function:



7.7.2.2 subroutine sequential_flux (real, dimension(toteq,npt) *flux*, real, dimension(nstate,npt) *wminus*, real, dimension(nstate,npt) *wplus*, real, dimension(toteq,npt) *uminus*, real, dimension(toteq,npt) *uplus*, real, dimension(3,npt) *jaminus*, real, dimension(3,npt) *japlus*, external *fluxfunction*, *nstate*, *npt*)

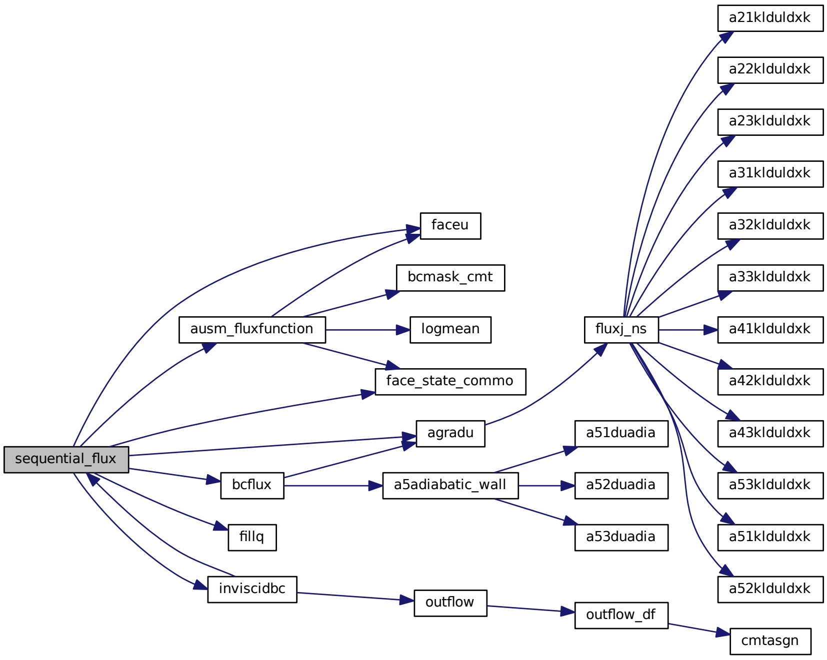
Calls two-point external fluxfunction $F^{\#}(U^{-},U^{+})$ at npt points Mostly intended to allow quantity-innermost volume flux functions to be used where needed for surface fluxes at boundary points, after $*$bc routines provide Dirichlet “rind” states in wplus and uplus.

Parameters

flux	Real (intent(out)) $F^{\#}(U^{-},U^{+})$ on output
wminus	Real (intent(in)) Primitive variables at one point. Usually w^{-} at interior nodes
wplus	Real (intent(in)) Primitive variables at the other point. Usually w^{+} at rind state or nodes of neighboring element
jaminus	Real (intent(in)) Mesh metrics at one point.
japlus	Real (intent(in)) Mesh metrics at the other point.
uminus	Real (intent(in)) Conserved variables one point. Usually U^{-} at interior nodes.
uplus	Real (intent(in)) Conserved variables at the other point. Usually U^{+} at rind state or nodes of neighboring element

External subroutine for $F^{\#}$. See fluxfn.f

Here is the call graph for this function:



7.8 utility functions for manipulating face data

7.9 structure for symmetric flux functions in split forms

7.10 flow field initialization routines

Functions/Subroutines

- subroutine cmtasgn (ix, iy, iz, e)

  Fill /NEKUSE/ and /NEKUSCMT/ common blocks from a single GLL node extends nekasgn to CMT-nek without affecting core routines.
- subroutine cmt_ics

  set initial values of conserved variables in U for CMT-nek
- subroutine cmtuic

  Fresh initialization of conserved variables from useric.

7.10.1 Detailed Description

7.10.2 Function/Subroutine Documentation

7.10.2.1 subroutine cmt_ics ( )

set initial values of conserved variables in U for CMT-nek

Over-engineered duplicate of setics in core nek5000. Calls cmtuic for a fresh start or my_full_restart for restart. cmtuic actually initializes the flow field through cmt-nek's own dedicated calls to useric. logs min and max of primitive variables as a sanity check in diagnostic I/O labeled "Cuvwpt," etc.

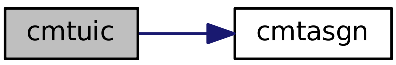
Here is the call graph for this function:

7.10.2.2 subroutine cmtuic ( )

Fresh initialization of conserved variables from useric.

Calls cmtasgn to interface with userbc, forms conserved variables from scalar primitive variables one grid point at a time, and fills U completely.

Here is the call graph for this function:



7.11 Thermodynamic state variables from conserved variables

Functions/Subroutines

- subroutine compute_primitive_vars (ilim)

  Compute primitive variables (velocity, thermodynamic state) from conserved unknowns U and store them in SOLN and CMTDATA.
- subroutine tdstate (e, energy)

  calls cmt_userEOS in the usr file. Compute thermodynamic state for element e from internal energy and density.
- subroutine poscheck (ifail, what)

  if positive posflags, write failure message and exit

7.11.1 Detailed Description

7.11.2 Function/Subroutine Documentation

7.11.2.1 subroutine compute_primitive_vars (integer *ilim*)

Compute primitive variables (velocity, thermodynamic state) from conserved unknowns U and store them in SOLN and CMTDATA.

Parameters

ilim	ilim is a flag for positivity checks. ilim==0 means `do not perform positivity checks."  ilim !=0 means`perform positivity checks and exit with a diagnostic dump if density, energy or temperature fall to zero or below at any GLL node."

Flags for density, energy and temperature positivity

Density positivity check.

Divide momentum by density to get velocity

Compute kinetic energy using vdot2/3

Compute internal energy. First, subtract volume-fraction-weighted kinetic energy from total energy.

Then, divide internal energy by density.

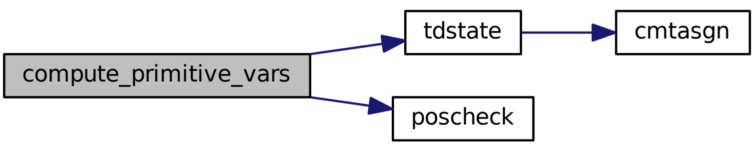
Compute density by dividing U1 by gas volume fraction. store in vtrans(:,jrho)

Check positivity of internal energy.

Compute thermodynamic state variables cv, T and p. Check temperature positivity using ifailt in /posflags/

call poscheck for each of the posflags and exit if ilim!=0 and any posflag>0. Nonzero posflags are set to the global element number where the first positivity failure on each MPI task was encountered.

Here is the call graph for this function:



7.11.2.2 subroutine tdstate (integer *e*, real, dimension(lx1,ly1,lz1) *energy*)

calls cmt_userEOS in the usr file. Compute thermodynamic state for element e from internal energy and density.

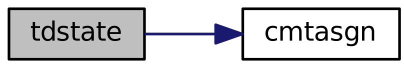
loop over GLL nodes in element e. Fill /NEKUSE/ and /nekuscmt/ by nekasgn and cmtasgn calls, respectively.

Compute thermodynamic state from scalars declared in /NEKUSE/ and /nekuscmt/ store state variables in temp,cv,cp,pres and asnd

Check temperature positivity

Fill SOLN and CMTDATA arrays one GLL node at a time from scalars in /NEKUSE/ and /nekuscmt/

Here is the call graph for this function:



Chapter 8

File Documentation

8.1 bc.f File Reference

Boundary condition routines.

Functions/Subroutines

- subroutine inviscidbc (flux)

  Determining rind state for Dirichlet boundary conditions.
- subroutine bcmask_cmt (bmsk)

  Mask to make sure Fsharp doesn't clobber boundary faces, where gs_op is null This routine intents to take a real array for all face points, bmask, and only zero out faces on boundaries. It is thus not limited to an array only of indicators.
- subroutine bcflux (flux, agradu, qminus)

  Determining IGU contribution to boundary flux. 0 for artificial viscosity, and strictly interior for physical viscosity.
- subroutine a5adiabatic_wall (eflx, f, e, dU, wstate)

  computes boundary flux for adiabatic wall in igu
- subroutine a51duadia (flux, f, ie, dU, wstate)

  same as A51 for volume flux (x-direction viscous flux of energy, but
- subroutine a52duadia (flux, f, ie, dU, wstate)

  same as A51 for volume flux (y-direction viscous flux of energy, but
- subroutine a53duadia (flux, f, ie, dU, wstate)

  same as A51 for volume flux (z-direction viscous flux of energy, but

8.1.1 Detailed Description

Boundary condition routines.

8.2 diffusive_cmt.f File Reference

routines for diffusive fluxes. Some surface. Some volume. All pain. Jacobians and other factorizations.

Functions/Subroutines

- subroutine br1auxflux (e, flux, ujump)

  add BR1 auxiliary flux $\frac{1}{2}\left(\mathbf{U}^{+}-\mathbf{U}^{-}\right)$ *to the gradient for a single element*

- subroutine imqqtu (ummcu, uminus, uplus)

  ummcu = $\mathbf{U}^{-} - \{\{\mathbf{U}\}\}$
- subroutine imqqtu_dirichlet (umubc, wminus, wplus)

  $\left(\mathbf{I} - 1/2QQ^T\right)\mathbf{U} \rightarrow \mathbf{U}^{-} - \mathbf{U}^{D}$ *on Dirichlet boundaries. Currently only modifies* $\mathbf{U}^{+}$ *in the case of walls.*
- subroutine agradu (flux, du, e, eq)

  Transforms the gradients of conserved variables $\nabla\mathbf{U}$ *to the viscous flux* $\mathbf{H}^{(d)}$ *in a single element for a single equation. Particular choice of viscous stress tensor is currently hardcoded for Navier-Stokes.*
- subroutine fluxj_ns (flux, gradu, e, eq)

  $\tau_{ij} = 2\mu\sigma_{ij} + \lambda\Delta\delta_{ij}$ *Navier-Stokes. uservp provides properties stored in SOLN. Implemented via maxima-generated code.*
- subroutine fluxj_evm (flux, du, e, eq)

  viscous flux jacobian for entropy viscosity Euler regularization of Guermond and Popov (2014) SIAM JAM 74(2) that do NOT overlap with the compressible Navier-Stokes equations (NS).
- subroutine half_iku_cmt (res, diffh, e)

  Compute the integrand $\mathbf{D}^T\mathbf{H}^d$ *of the weak-form volume integral and store it in res1 one element per call.*
- subroutine compute_transport_props

  Fill vdiff with transport properties. Hardcoded indices for Navier-Stokes Used for both artificial and physical viscosities.
- subroutine a51kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.
- subroutine a52kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.
- subroutine a53kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.
- subroutine a21kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.
- subroutine a22kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.
- subroutine a23kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.
- subroutine a31kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.
- subroutine a32kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.
- subroutine a33kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.
- subroutine a41kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.
- subroutine a42kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.
- subroutine a43kldukdxk (flux, dU, ie)

  WRITE OUT NOTATION.

8.2.1 Detailed Description

routines for diffusive fluxes. Some surface. Some volume. All pain. Jacobians and other factorizations.

8.3 drive1_cmt.f File Reference

high-level driver for CMT-nek

Functions/Subroutines

- subroutine cmt_nek_advance

  Branch from subroutine nek_advance in core/drive1.f Advance CMT-nek one time step within nek5000 time loop.
- subroutine compute_rhs_and_dt

  Compute right-hand-side of the semidiscrete conservation law Store it in res1.
- subroutine set_tstep_coef

  Compute coefficients for Runge-Kutta stages ***[TVDRK]****}.*
- subroutine cmt_flow_ics

  This subroutine must only be called after a restart. It copies arrays that nek5000 reads from SLN files into their corresponding slots in CMTDATA. vx stores U(:,2,:), x-momentum. vy stores U(:,3,:), y-momentum. vz stores U(:,4,:), z-momentum. pr stores U(:,1,:), fluid density The T array stores U(:,5,:), fluid total energy.
- subroutine **print_cmt_timers**
- subroutine **init_cmt_timers**

8.3.1 Detailed Description

high-level driver for CMT-nek

8.3.2 Function/Subroutine Documentation

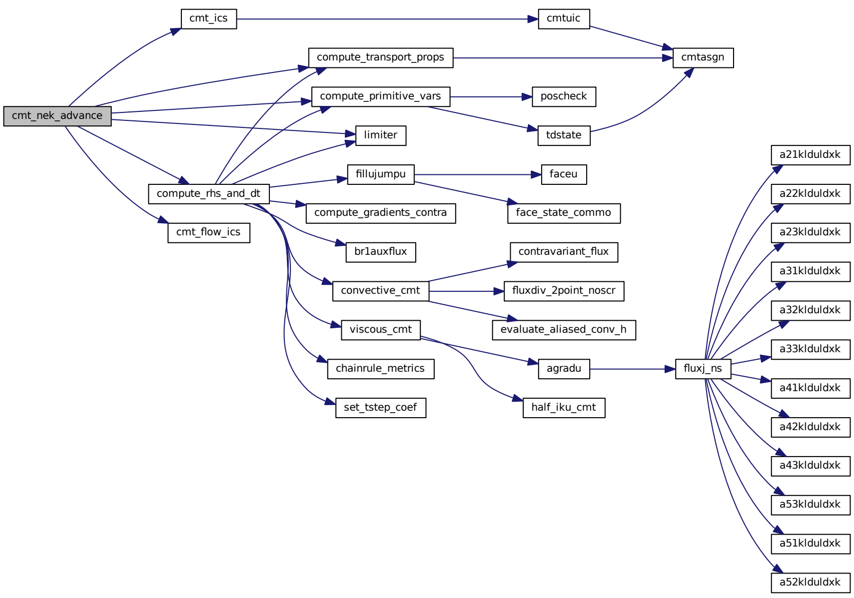
8.3.2.1 subroutine cmt_nek_advance ( )

Branch from subroutine nek_advance in core/drive1.f Advance CMT-nek one time step within nek5000 time loop.

Initialization calls

Runge-Kutta loop

Here is the call graph for this function:



8.3.2.2 subroutine compute_rhs_and_dt ( )

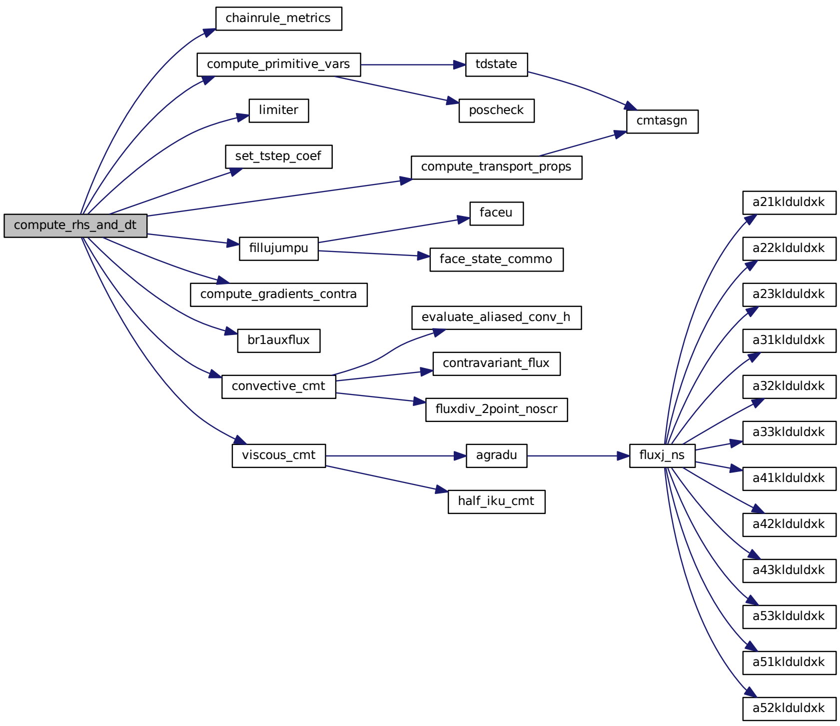
Compute right-hand-side of the semidiscrete conservation law Store it in res1.

Restrict via $\mathbf{E}$ to get primitive and conserved variables on interior faces $\mathbf{U}^-$ and neighbor faces $\mathbf{U}^+$; store in CMT-SURFLX

res1+= $\oint \mathbf{H}^{c*} \cdot \mathbf{n} dA$ on face points

res1+= $\int_\Gamma \{\{\mathbf{A}\nabla\mathbf{U}\}\} \cdot [v]\, dA$

Here is the call graph for this function:



8.4 drive2_cmt.f File Reference

mid-level initialization drivers. Not long for this world.

Functions/Subroutines

- subroutine nek_cmt_init

  This routine was intended to integrate more carefully with nek5000's gs_op ifheat was supposed to trigger initialization of Paul Fischer's implementations of DG operators K, G and G^T, but we never finished keeping up with their work. Right now it's just a wrapper for setup_cmt_commo.
- subroutine izero8 (a, n)

  vector routine to zero out kind=8 integers.
- subroutine limiter

  positivity-preserving limiters. Adjusts conserved variables in U for to ensure that density, pressure and internal energy are positive. Follows Zhang & Shu (2010) JCP 229. We did get Lv & Ihme's (2015) entropy-bounded discontinuous Galerkin (EBDG) limiter working, but this is only for perfect gases.
- real function logmean (l, r)

Ismail & Roe's (2009) version of the logarithmic mean for all possible pairs of values, including equal values.

8.4.1 Detailed Description

mid-level initialization drivers. Not long for this world.

8.5 driver3_cmt.f File Reference

routines for primitive variables, usr-file interfaces and properties. Also initializes flow field.

Functions/Subroutines

- subroutine compute_primitive_vars (ilim)

  Compute primitive variables (velocity, thermodynamic state) from conserved unknowns U and store them in SOLN and CMTDATA.
- subroutine tdstate (e, energy)

  calls cmt_userEOS in the usr file. Compute thermodynamic state for element e from internal energy and density.
- subroutine cmtasgn (ix, iy, iz, e)

  Fill /NEKUSE/ and /NEKUSCMT/ common blocks from a single GLL node extends nekasgn to CMT-nek without affecting core routines.
- subroutine cmt_ics

  set initial values of conserved variables in U for CMT-nek
- subroutine cmtuic

  Fresh initialization of conserved variables from useric.
- subroutine poscheck (ifail, what)

  if positive posflags, write failure message and exit

8.5.1 Detailed Description

routines for primitive variables, usr-file interfaces and properties. Also initializes flow field.

8.6 eqnsolver_cmt.f File Reference

Routines for entire terms on RHS. Mostly volume integrals.

Functions/Subroutines

- subroutine viscous_cmt (e, eq)

  Volume integral for diffusive terms.
- subroutine igtu_cmt (qminus, ummcu, hface)

  Computes $G^T U$, the volume integral of $[[u]] \cdot \{\{\nabla v\}\}$, and increments res1 with it.
- subroutine convective_cmt (e)

  Evaluates inviscid volume terms for all toteq equations in two-point split form (Equation∼shortvol}and adds them to res1(:,:,:,e,:).
- subroutine **fluxdiv_2point_slow** (res, e, ja)
- subroutine **fluxdiv_2point_scr** (res, fcons, e, ja)
- subroutine fluxdiv_2point_noscr (res, fcons, e, ja)

  Evaluates the two-point split form (Equations∼splitr} through splitt}) of the volume integral $\int v \nabla \cdot \mathbf{H}^c dV$ and the discontinuous surface flux $\oint v \mathbf{H}^c \cdot \mathbf{n} dA$ for the inviscid flux function in a single element.

- subroutine fluxdiv_strong_contra (e)

  Computes $\int v\nabla\cdot\mathbf{H}^c dV$ in aliased strong form for element e, and increments res1 with it.
- subroutine evaluate_aliased_conv_h (e)

  Evaluates consistent (i.e. $F^{\#}(U_{i,j,k},U_{l,j,k})=H_{(eq),i}, i=l$) flux function at all GLL nodes and stores it in convh.
- subroutine fluxdiv_dealiased_weak_chain (e)

  $(\nabla v)\cdot\mathbf{H}^c=\mathscr{I}^\intercal\mathbf{D}^\intercal\cdots$ *for equation eq, element e*
- subroutine **fluxdiv_weak_chain** (e)
- subroutine contravariant_flux (frst, fxyz, ja, nel)

  Transforms consistent (i.e. $F^{\#}(U_{i,j,k},U_{l,j,k})=H_{(eq),i}, i=l$) flux for one conserved variable to the contravariant frame by contracting H_i with J diag $\left(\partial r_j/\partial x_i\right)$.
- subroutine **compute_forcing** (e, eq_num)
- subroutine **cmtusrf** (e)

8.6.1 Detailed Description

Routines for entire terms on RHS. Mostly volume integrals.

8.7 face.f File Reference

low-level initialization drivers. Eventually to be superceded by nek5000 core DG handles and operators.

Functions/Subroutines

- subroutine **iface_vert_int8cmt** (nx, ny, nz, fa, va, jz0, jz1, nel)
- subroutine **setup_cmt_gs** (dg_hndl, nx, ny, nz, nel, melg, vertex, gnv, gnf)
- subroutine **setup_cmt_commo**
- subroutine **cmt_set_fc_ptr** (nel, nx, ny, nz, nface, iface)
- subroutine **full2face_cmt** (nel, nx, ny, nz, iface, faces, vols)
- subroutine **add_face2full_cmt** (nel, nx, ny, nz, iface, vols, faces)

8.7.1 Detailed Description

low-level initialization drivers. Eventually to be superceded by nek5000 core DG handles and operators.

8.8 fluxfn.f File Reference

Riemann solvers, other rocflu miscellany and two-point fluxes.

Functions/Subroutines

- subroutine **kgrotfluxfunction** (ntot, nm, rl, ul, vl, wl, pl, al, tl, rr, ur, vr, wr, pr, ar, tr, flx, el, er)
- subroutine ausm_fluxfunction (ntot, nx, ny, nz, nm, fs, rl, ul, vl, wl, pl, al, tl, rr, ur, vr, wr, pr, ar, tr, flx, el, er)

  Computes inviscid numerical surface flux from AUSM+ Riemann solver.
- subroutine **centralinviscid_fluxfunction** (ntot, nx, ny, nz, fs, ul, pl, ur, pr, flx)
- subroutine **llf_euler** (flx, ul, ur, wl, wr, nrm, dum)
- subroutine **llf_euler_vec** (wminus, uplus, flux, nstate)
- subroutine **kennedygruber** (flx, ul, ur, wl, wr, jal, jar)
- subroutine **kepec_ch** (flx, ul, ur, wl, wr, jal, jar)
- subroutine **kennedygruber_vec** (z, flux, nstate, nflux)

- subroutine **trivial**
- subroutine **rhoe_to_e** (fatface, nf, ns)
- subroutine **parameter_vector_vol** (z, zt, ut, e, idum)
- subroutine **kepec_duplicated** (wminus, wplus, flux)

8.8.1 Detailed Description

Riemann solvers, other rocflu miscellany and two-point fluxes.

8.9 intpdiff.f File Reference

interpolation and differentiation routines not already provided by nek5000

Functions/Subroutines

- subroutine compute_gradients (e)

  Compute gradients of conserved variables (WITHOUT volume fraction weighting) for a single element. Store them in gradu in /CMTGRADU/. Uses legacy chain-rule metrics in DXYZ (rxm1, etc.) and NOT freestream-preserving metrics.
- subroutine set_dealias_face

  Fills arrays in /FACEWZ/ with Gauss-Legendre quadrature weights on the fine grid for dealiasing surface integrals.
- subroutine cmt_metrics (istp)

  compute freestream-preserving metrics Ja^i for transforming fluxes F to $\tilde{F}$ in a contravariant frame according to Kopriva (2006) Follows methodology in FLUXO (github.com/project-fluxo/fluxo.git) Basically, isoparametric mappings are not freestream preserving. DGSEM needs metrics computed from low-order geometry interpolated up to polynomial order (when lx1>ngeo) OR "dealiased" metrics (when ngeo>lx1)
- subroutine **xyztriv** (xl, yl, zl, nxl, nyl, nzl, e)
- subroutine **get_int_gll2gll** (ip, mx, md)
- subroutine **gen_int_gll2gll** (jgl, jgt, mp, np, w)
- subroutine **proj_legmodal** (promat, nin, nout)
- subroutine **discard_rows** (trunc, matrix, nsmall, nlarge)
- subroutine **vandermonde_legendre** (v, z, nx)
- subroutine **gradm11_t** (grad, uxyz, csgn, e)
- subroutine **gradm11_t_contra** (grad, uxyz, csgn, e)
- subroutine **gradm1_t** (u, ux, uy, uz)

- subroutine compute_gradients_contra (e)

  Compute gradients of conserved variables (WITHOUT volume fraction weighting) for a single element. Store them in gradu in /CMTGRADU/. Uses freestream-preserving metrics that cmt_metrics computes and stores in rx.

- subroutine chainrule_metrics (istp)

  Keep chain-rule metrics as a stop-gap until cmt_metrics works on deformed elements. Also fills jface with jacobian of mesh on mesh faces, w2m1 with quadrature weights, and dstrong with derivative matrix and appropriate weights for strong-form surface integrals.

8.9.1 Detailed Description

interpolation and differentiation routines not already provided by nek5000

8.9.2 Function/Subroutine Documentation

8.9.2.1 subroutine chainrule_metrics (*istp*)

Keep chain-rule metrics as a stop-gap until cmt_metrics works on deformed elements. Also fills jface with jacobian of mesh on mesh faces, w2m1 with quadrature weights, and dstrong with derivative matrix and appropriate weights for strong-form surface integrals.

dstrong in Equation~dstrong} used to evaluate discontinuous surface flux on the fly.

8.9.2.2 subroutine compute_gradients (integer *e*)

Compute gradients of conserved variables (WITHOUT volume fraction weighting) for a single element. Store them in gradu in /CMTGRADU/. Uses legacy chain-rule metrics in DXYZ (rxm1, etc.) and NOT freestream-preserving metrics.

Parameters

e	e is intent(in). It is the index for the current element in the element loop in compute_rhs_and-_dt

Divide conserved variables by volume fraction

Differentiate in the reference element. local_grad routines are in core/navier5.f

Convert d/dr to d/dx via chain-rule metrics.

8.9.2.3 subroutine compute_gradients_contra (integer *e*)

Compute gradients of conserved variables (WITHOUT volume fraction weighting) for a single element. Store them in gradu in /CMTGRADU/. Uses freestream-preserving metrics that cmt_metrics computes and stores in rx.

Parameters

e	e is intent(in). It is the index for the current element in the element loop in compute_rhs_and-_dt

Divide conserved variables by volume fraction

Differentiate conserved variables in the reference element. local_grad routines are in core/navier5.f

8.9.2.4 subroutine set_dealias_face ( )

Fills arrays in /FACEWZ/ with Gauss-Legendre quadrature weights on the fine grid for dealiasing surface integrals.

Gauss-Legendre quadrature weights. ZWGL lives in core/speclib.f

Tensor product in two dimensions for a face of a 3D element.

faces are just edges in 1D. No tensor product

8.10 outflow_bc.f File Reference

Dirichlet states for outflow boundary conditions wrapper for other BC routines. Just one for now. More to come.

Functions/Subroutines

- subroutine outflow (f, e, wminus, wplus, uminus, uplus, nvar)
- subroutine outflow_df (f, e, wm, wp, um, up, nvar)

Dolejsi & Feistauer (2015) Section 8.3.2.2. Very rudimentary "physical" boundary conditions. Also encountered in Hartmann & Houston (2006). A poor default.

- subroutine **outflow_rflu** (nvar, f, e, facew, wbc)
- subroutine **bcondoutflowperf** (bcOpt, pout, sxn, syn, szn, cpgas, mol, rho, rhou, rhov, rhow, rhoe, press, rhob, rhoub, rhovb, rhowb, rhoeb)

8.10.1 Detailed Description

Dirichlet states for outflow boundary conditions wrapper for other BC routines. Just one for now. More to come.

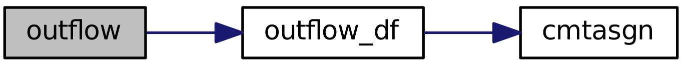
8.10.2 Function/Subroutine Documentation

8.10.2.1 subroutine outflow (integer *f*, integer *e*, real, dimension(nvar,lx1∗lz1) *wminus*, real, dimension(nvar,lx1∗lz1) *wplus*, real, dimension(toteq,lx1∗lz1) *uminus*, real, dimension(toteq,lx1∗lz1) *uplus*, integer *nvar*)

Parameters

f	face index from 1 to 2∗ldim
e	element index
nvar	number of primitive variables
wminus	primitive variables from flow solution (dimension(nvar,lx1∗lz1),intent(in))
wplus	external dirichlet state's primitive variables (dimension(nvar,lx1∗lz1),intent(out))
uminus	conserved variables from flow solution (dimension(toteq,lx1∗lz1),intent(in))
uplus	external dirichlet state's conserved variables (dimension(toteq,lx1∗lz1),intent(out))

Here is the call graph for this function:



8.11 step.f File Reference

time stepping and mesh spacing routines

Functions/Subroutines

- subroutine **setdtcmt**
- subroutine **mindr** (mdr, diffno)
- real function **dist2** (x1, y1, x2, y2)
- subroutine **compute_grid_h** (h, x, y, z)
- subroutine **glinvcol2max** (col2m, a, b, n, s)
- subroutine **glsqinvcolmin** (col2m, a, b, n, s)
- subroutine **compute_mesh_h** (h, x, y, z)

8.11.1 Detailed Description

time stepping and mesh spacing routines

8.12 surface_fluxes.f File Reference

Routines for surface terms on RHS.

Functions/Subroutines

- subroutine fillujumpu

  overwrite beginning of /CMTSURFLX/ with $-[[\mathbf{U}]]$ for flux of auxiliary variable in the viscous flux of Bassi and Rebay computed in br1auxflux.
- subroutine fluxes_full_field_kg

  Restrict and copy face data and compute inviscid numerical flux $\oint \mathbf{H}^{c*} \cdot \mathbf{n} dA$ on face points. This particular wrapper is for the symmetric flux of Kennedy and Gruber.
- subroutine faceu (ivar, yourface)

  Restrict and copy face data for conserved variables U_{ivar}. Wraps full2face_cmt for each element; a single call to full2face with U will not work because element varies with the outermost index of the u array.
- subroutine fillq (jvar, field, wminus, yourface)

  Restrict and copy face data for one full field and store it in index jvar in wminus.
- subroutine dg_face_avg (mine, nf, nstate, handle)

  Overwrite values stored at points on faces with the average with its values in the neighboring face without duplication. Replaces $mine$ with $\{\{mine\}\}$.
- subroutine face_state_commo (mine, yours, nf, nstate, handle)

  Sends face values v^- stored in “mine” to the neighbor that shares that face and copies the neighbor's values v^+ at each face into “yours.”.
- subroutine avg_and_jump (avg, jump, scratch, nf, nstate, handle)

  Overwrites w^- at interior face points stored in avg with $\{\{w\}\}$. jump gets filled with $[[w]]$.
- subroutine **face_flux_commo** (flux1, flux2, nf, neq, handle)
- subroutine sequential_flux (flux, wminus, wplus, uminus, uplus, jaminus, japlus, fluxfunction, nstate, npt)

  Calls two-point external fluxfunction $F^{\#}(U^-,U^+)$ at npt points Mostly intended to allow quantity-innermost volume flux functions to be used where needed for surface fluxes at boundary points, after $*$bc routines provide Dirichlet “rind” states in wplus and uplus.
- subroutine **inviscidflux** (wminus, wplus, flux, nstate, nflux)
- subroutine **surface_integral_full** (vol, flux)
- subroutine **diffh2graduf** (e, eq, graduf)
- subroutine **diffh2face** (e, eq, diffhf)
- subroutine **igu_cmt** (flxscr, gdudxk, wminus)
- subroutine **igu_dirichlet** (flux, agradu)
- subroutine **br1primary** (flux, gdudxk)
- subroutine **agradu_normal_flux** (flux, graduf)
- subroutine **br1bc** (flux)
- subroutine **bcflux_br1** (flux, f, e)
- subroutine **strong_sfc_flux** (flux, vflx, e, eq)
- subroutine **fluxes_full_field_chold**
- subroutine **fluxes_full_field_old**
- subroutine **inviscidfluxrot** (wminus, wplus, flux, nstate, nflux)
- subroutine gtu_wrapper (fatface)

8.12.1 Detailed Description

Routines for surface terms on RHS.

8.13 wall_bc.f File Reference

Dirichlet states for wall boundary conditions.

Functions/Subroutines

- subroutine **wallbc2** (nstate, f, e, facew, wbc)
- subroutine **wallbc_inviscid** (f, e, wminus, wplus, uminus, uplus, nvar)
- subroutine **reflect_rind** (f, e, wm, wp, um, up, nvar)
- subroutine **slipwall_rflu** (nvar, f, e, facew, wbc, fluxw)
- subroutine **rflu_setrindstateslipwallperf** (cpGas, mmGas, nx, ny, nz, rl, ul, vl, wl, fs, pl)

8.13.1 Detailed Description

Dirichlet states for wall boundary conditions.

Index