bcg.h Documentation

Haochen(Langford) Huang

May 28, 2024

version:2.0

1 Introduction and Background

bcg.h file contains two functions: *tracer_fluxes* and *advection* whose major purpose is to construct a solver for advective equation:

$$\frac{\partial \Phi}{\partial t} + (\mathbf{u} \cdot \nabla)\Phi = 0 \tag{1}$$

where Φ is the scalar and $\mathbf{u}$ is the velocity. The discrete time formulation of equation **??** reads:

$$\frac{\Phi^{n+1} - \Phi^n}{\Delta t} + \mathbf{A}^{n+\frac{1}{2}} = 0 \tag{2}$$

Where $\mathbf{A}^{n+\frac{1}{2}}$ is the abbreviation of advection term. With the conservative constraints $\nabla \cdot \mathbf{u} = 0$, the integral form reads:

$$\int_\Gamma A^{n+\frac{1}{2}} = \int_\Gamma [(\mathbf{u} \cdot \nabla)\Phi]^{n+\frac{1}{2}} = \int_\Gamma [\nabla \cdot (\mathbf{u}\Phi)]^{n+\frac{1}{2}} = \int_{\partial\Gamma} (\mathbf{u}^{n+\frac{1}{2}} \cdot \mathbf{n})\Phi^{n+\frac{1}{2}} \tag{3}$$

where $\mathbf{n}$ represents the normal direction of cell interface. For cartesian cell employed in Basilisk, the calculation turns out to be

$$\Delta A^{n+\frac{1}{2}} = \sum_d s_d u_{f,d}^{n+\frac{1}{2}} \Phi_{f,d}^{n+\frac{1}{2}} \tag{4}$$

Note in this documentation, the subscript f denotes face value stored in staggered mesh[**1965˙Harlow**], otherwise it is cell-average value stored in center of the cell. Subscript d is the notation of specific face (e, w, n, s for 2D and e, s, n, s, f, b for 3D) for a single cell. In addition, readers may found algebraic notation without d such as u_f in the upcoming discussion which refers to the same thing. This is because in the most cases d is used to addressed spatial relationship between face and cell from cell-perspective, such relation shall vanish from face-perspective and so does d.

The following discussions are conducted under 2D condition and results of 3D is easily to obtain accordingly. s_d represents sign function which reads

$$s_d = \begin{cases} 1 & d = e, n \\ -1 & d = w, s \end{cases} \tag{5}$$

The problem now becomes how to obtain $u_f^{n+1/2}$ and $\Phi_f^{n+1/2}$. Basilisk employs BCG scheme to tackle such issue. The scheme, which is originally named by three authors[**1989˙Bell**], stems from the algorithm intending to apply second order Godunov method to incompressible flow. It obtains value at $n + \frac{1}{2}$ not by averaging but by considering the Taylor series, for each cell:

$$\tilde{\Phi}_{f,d}^{n+\frac{1}{2}} = \Phi^n + s_d \frac{\Delta}{2} \frac{\partial \Phi^n}{\partial x_d} + \frac{\Delta t}{2} \frac{\partial \Phi^n}{\partial t} + O(\Delta^2, \Delta t^2) \tag{6}$$

Where $x_d = x$ for $d = e, w$ and $x_d = y$ for $d = n, s$. Replacing $\frac{\partial \Phi^n}{\partial x_d}$ with Euler equation **??** yielding

$$\tilde{\Phi}_{f,d}^{n+\frac{1}{2}} = \Phi^n + [s_d \frac{\Delta}{2} - \frac{\Delta t}{2} u_d] \frac{\partial \Phi^n}{\partial x_d} - \frac{\Delta t}{2} u_o \frac{\partial \Phi^n}{\partial x_o} \tag{7}$$

where u_d is the cell-centered value on d direction (i.e. u_x for $d = e, w$, u_y for $d = n, s$). Subscript o represents directions other than d. Since aiming equation **??** is advection-only, a simple upwind scheme is therefore employed[**2000˙Martin**, **1998˙Martin**] to adapt each term in equation **??**:

$$\tilde{\Phi}_{f,d}^{n+\frac{1}{2}} = \Phi^n + [s_d\frac{\Delta}{2} - max(\frac{\Delta t}{2}u_d, 0)]\frac{\partial \Phi^n}{\partial x_d} - \frac{\Delta t}{2}u_o\overline{\frac{\partial \Phi^n}{\partial x_o}} \tag{8}$$

$$\overline{\frac{\partial \Phi^n}{\partial x_o}} = \begin{cases} (\Phi^n[0] - \Phi^n[-1])/\Delta & if\ u_o > 0 \\ (\Phi^n[1] - \Phi^n[0])/\Delta & if\ u_o < 0 \end{cases} \tag{9}$$

If not additionally declared, $\frac{\partial \Phi^n}{\partial x_d}$ is computed with central scheme. Given four face values from each cell, each face now has two values from its neighbor cell which are left value and right value in traditional Godunov method. Unlike the traditional method which obtains the final face value $\Phi_{f,d}$ by solving Riemann problem, the upwind scheme is again applied to determine $\Phi_{f,d}$ in current algorithm (the value is counted from each face hence the subscript d is omitted.)

$$\Phi_f^{n+1/2}[0] = \begin{cases} \tilde{\Phi_f}[0] & u_f^{n+1/2} < 0 \\ \tilde{\Phi_f}[-1] & u_f^{n+1/2} > 0 \\ \frac{1}{2}(\tilde{\Phi_f}[0] + \tilde{\Phi_f}[-1]) & u_f^{n+1/2} = 0 \end{cases} \tag{10}$$

where u_f is velocity on the same face and marker $[0], [-1]$ indicate the cell from which the face value is obtained. In mesh system of Basilisk, for face located at $(i - 1/2, j)$ the $[0]$ (resp. $[-1]$) represents cell positioned at (i, j) (resp. $(i - 1, j)$). It is worth mentioning that the marker in current equation is associated with the direction. For example, if one calculates face value $\Phi_f^{n+1/2}$ on x direction, the marker in equation **??** indicates the relationship on x direction while those in equation **??** refers to relation on y direction. Above discussion is the original algorithm on the paper, compensation is made in real code which shall be introduced in the following sections.

2 Functions

2.1 tracer_fluxes

Given face velocity $u_f^{n+1/2}$, the face flux $u_f^{n+\frac{1}{2}}\Phi_f^{n+\frac{1}{2}}$ is computed and stored on staggered mesh.

2.1.1 Parameters

Name	Data type	Status	Option	Representation (before/after)
f	scalar	unchanged	compulsory	Φ^n
uf	face vector	unchanged	compulsory	$u_f^{n+\frac{1}{2}}$
flux	face vector	**output**	compulsory	$u_f^{n+\frac{1}{2}}\Phi_f^{n+\frac{1}{2}}$
dt	double	unchanged	compulsory	Δt
src	scalar	unchanged	compulsory	a^n

2.1.2 Worth Mentioning Details

As discussed previously, specific face value on staggered mesh[**1965˙Harlow**] can be located by its spatial relation with corresponding cell. Take cell at (i, j) as an example, figure **??** demonstrates notation of each face. Moreover such value can be iterated by calling macro $foreach\_face()$ in Basilisk. Thanks to this feature, the code can be constructed from face-perspective starting from equation **??** rather than iterating every cell and compare two values for single faces.

Notably, different from original algorithm, two compensations are made in performing equation **??** and equation **??** respectively. For the former one, the condition where $u_f^{n+1/2} = 0$ merges into the second condition, the filter now reads:

$$\Phi_f^{n+1/2}[0] = \begin{cases} \tilde{\Phi_f}[0] & u_f^{n+1/2} < 0 \\ \tilde{\Phi_f}[-1] & u_f^{n+1/2} \geq 0 \end{cases} \tag{11}$$

For the latter one, u_d is replaced by face value $u_f[]$ for simplicity in boundary condition settings. Consequently the upwind filter (i.e. the $max()$) is no longer needed in the second term of equation **??** since upwind scheme is automatically met owing to upwind selection originally conducted by equation **??**.

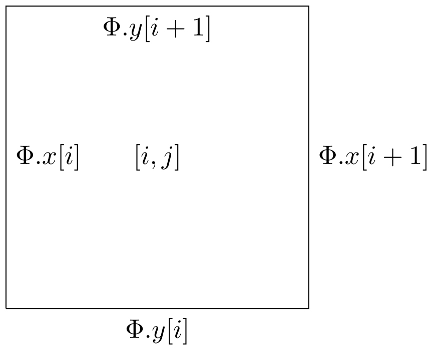

Figure 1: Example of staggered mesh.

Given $u_f^{n+1/2}$, the procedure to compute face flux now becomes:

1. Identify the upstream direction based on $u_f^{n+1/2}$.
2. Choose corresponding cell to compute face flux based on equation **??** and upstream direction obtained in previous step.
3. According to equation **??** **??**, add contribution from each direction and multiply the result by $u_f^{n+1/2}$ to obtain $u_f^{n+1/2}\Phi_f^{n+1/2}$.

Yet another detail worth mentioning is the source term a^n in current function. This is indeed preparation for solving non-Euler equation such as NS equation in 'centered.h'. Albeit compulsory here, such term is overall optional in the final function **advection** which shall be introduced in the upcoming section.

2.1.3 Program Workflow

Starting Point
input:
$f = \Phi^n$ $uf = u_f^{n+\frac{1}{2}}$
$flux$(empty) dt=Δt
$src = \mathbf{g}^n$
gradient:
$g = \nabla f = \nabla \Phi$

```
void tracer_fluxes (scalar f,
                    face vector uf,
                    face vector flux,
                    double dt,
                    (const) scalar src)
{
  vector g[];
  gradients ({f}, {g});
```

compute un[a]:
$un=\frac{u_{f,d}^{n+1/2}\Delta t}{fm.x[i]\Delta}$
Identification of upstream direction
The notation of corresponding cell i is identified according to equation **??**. $s = s_d$

[a]A compensation is made here, see **??** for detailed information

```
  foreach_face() {
    double un = dt*uf.x[]/(fm.x[]*Delta +
    ↪  SEPS), s = sign(un);
    int i = -(s + 1.)/2.;
```

Contribution of Each Direction
Main direction
$f2 = \Phi^n[i] + \frac{\Delta}{2}[s_d - \frac{\Delta t}{\Delta} u_{f,d}^{n+1/2}]\frac{\partial \Phi^n}{\partial x_d}[i] + \frac{\Delta t}{2}\frac{a^n[-1]+a^n[0]}{2}$
Traversal direction
compute u_o in equation ??:
vn(*wn*)=$(u_{f,o}^{n+1/2}[i,0] + u_{f,o}^{n+1/2}[i,1])/(fm.e[i,0] + fm.e[i,1])$
component of other direction:
The gradient $\frac{\partial \Phi^n}{\partial x_o}$ is first computed by upwind scheme based on direction of u_o according to equation **??**
fyy(*fzz*)= $\frac{\partial \Phi^n}{\partial x_o}$
f2 -= $\frac{\Delta t}{2} u_o \frac{\partial \Phi^n}{\partial x_o}$

```
    double f2 = f[i] + (src[] + src[-1])*dt/4. + s*(1. - s*un)*g.x[i]*Delta/2.;
    #if dimension > 1
    if (fm.y[i] && fm.y[i,1]) {
      double vn = (uf.y[i] + uf.y[i,1])/(fm.y[i] + fm.y[i,1]);
      double fyy = vn < 0. ? f[i,1] - f[i] : f[i] - f[i,-1];
      f2 -= dt*vn*fyy/(2.*Delta);
    }
    #endif
    #if dimension > 2
    if (fm.z[i] && fm.z[i,0,1]) {
      double wn = (uf.z[i] + uf.z[i,0,1])/(fm.z[i] + fm.z[i,0,1]);
      double fzz = wn < 0. ? f[i,0,1] - f[i] : f[i] - f[i,0,-1];
      f2 -= dt*wn*fzz/(2.*Delta);
    }
    #endif
```

Assemble
assemble the final result, current direction is d
flux.d=$u_{f,d}^{n+\frac{1}{2}} \Phi_{f,d}^{n+\frac{1}{2}}$
after the $foreach\_face$ macro, the face flux on every face is obtained.

```
    flux.x[] = f2*uf.x[];
  }
}
```

2.2 advection

2.2.1 Parameters

Name	Data type	Status	Option	Representation (before/after)
tracers	scalar*	update	compulsory	$\Phi^n / \ \Phi^n - \Delta t A^{n+\frac{1}{2}}$
u	face vector	unchanged	compulsory	$u_d^{n+\frac{1}{2}}$
dt	double	unchanged	compulsory	Δt
src	scalar*	unchanged	optional	$\mathbf{g}^n$

2.2.2 Worth Mentioning Details

The input of this function (*tracers*,*src*) can be *vector* type, e.g. **u**, where components of each direction is deemed as *scalar* type data then is finally assembled as a *vector*.

Another thing needs to be concerned is the direction of the normal unity associates with coordinate axis. Positive direction of coordinate remains unchanged will face normal direction varies with respect to current cell. Take Fig.**??** as an example.

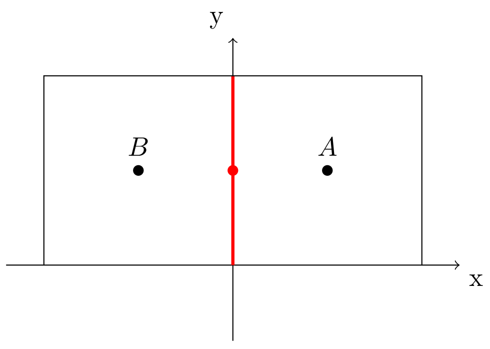

Figure 2: Face normal example.

For cell A, direction of highlighted face is opposite to coordinate while positive for that of cell B. For the sake of clarity and convenience, all value involved compute in function **tracer_fluxes** take default positive direction. Which means for further computation for $\sum_{i=d} u_d^{n+\frac{1}{2}} \Phi_d^{n+\frac{1}{2}}$ of cell A, the value added at highlighted face should take negative value reads

$$u_d^{n+\frac{1}{2}} \Phi_d^{n+\frac{1}{2}}(i) + u_d^{n+\frac{1}{2}} \Phi_d^{n+\frac{1}{2}}(i+1) = -flux.x[i] + flux.x[i+1] \tag{12}$$

2.2.3 Program Workflow

Starting Point
input:
tracers $= \Phi^n$ *u* $= u_d^{n+\frac{1}{2}}$
dt$=\Delta t$ *src* $= \mathbf{g}^n$

```
  struct Advection {
    scalar * tracers;
    face vector u;
    double dt;
    scalar * src; // optional
  };
  void advection (struct Advection p)
  {
    scalar * lsrc = p.src;
    if (!lsrc)
      for (scalar s in p.tracers)
        lsrc = list_append (lsrc,
         ↪  zeroc);
    assert (list_len(p.tracers) ==
     ↪  list_len(lsrc));
```

$$\downarrow$$

Fluxes Compute
Traversal each elements in *tracers* (if *tracers* is vector, then this step traversal component on every direction)
computation:
flux $= \Phi_d^{n+\frac{1}{2}} u_d^{n+\frac{1}{2}}$

```
scalar f, src;
for (f,src in p.tracers,lsrc) {
  face vector flux[];
  tracer_fluxes (f, p.u, flux, p.dt,
   ↪  src);
```

$$\downarrow$$

Update
tracers(updated) $=$
$\Phi^{n+1} = \Phi^n - \Delta t A^{n+\frac{1}{2}}$
$= \Phi^n - \Delta t \sum u_d^{n+\frac{1}{2}} \Phi_d^{n+\frac{1}{2}} = f[] +$
$\Delta t \sum_{each\_di}(flux.d[i] - flux.d[i-1])$

```
  #if !EMBED
      foreach()
        foreach_dimension()
          f[] += p.dt*(flux.x[] -
           ↪  flux.x[1])/(Delta*cm[]);
  #else // EMBED
      update_tracer (f, p.u, flux,
       ↪  p.dt);//This is a function
       ↪  that induced by embed.h which
       ↪  conducts same procedure with
       ↪  special care taken for embed
       ↪  boundary.
  #endif // EMBED
    }

    if (!p.src)
      free (lsrc);
  }
```

A Calculation of Face Centered Normal Velocity

It can be seen from **??**, the computation of face centered normal velocity $u_d^{n+\frac{1}{2}}$ is of great importance to construct advection term. The detailed procedure of extrapolation is shown in documentation of **centered.h**. The output of such variable follows a similar step as equation **??** but without pressure term since the face centered velocity will be projected with an edge-centered projection. This additional projection ensures the feature of conservative of corresponding method.

When it comes to the resolution of NS equation, Φ in governing equation **??** is the cell-centered vector $\mathbf{u}^n$. According to Martin[**martin2000cell**], normal components calculated by equation **??** can simply be replaced by $u_d^{n+\frac{1}{2}}$, which means we only need to compute tangential fluxes. However, after series of tests, reuse of normal fluxes will lead to unstable at sharp angles. Thus all components is recomputed in Basilisk[**popinet2003gerris**].