bcg.h Documentation

Haochen(Langford) Huang

April 23, 2024

version:1.0

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 1 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 in BCG[1, 4] scheme (and is the reason why the name of this file called 'bcg.h'), and is intergrated within the controlled volume

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

The second step above is achieved by combining divergence free constraints $\nabla \cdot \mathbf{u} = 0$ and the $\mathbf{n}$ represents the normal direction of cell interface. For quadtree or octree cells, the overall calculation turns out to be

$$\Delta A^{n+\frac{1}{2}} = \sum_{i=d} u_d^{n+\frac{1}{2}} \Phi_d^{n+\frac{1}{2}} \tag{4}$$

where u_d is the component of $\mathbf{u}$ on normal direction of interface, $Delta$ is the length of the cell and Φ_d is the corresponding value at the interface. Then the critical is how to obtain $\Phi_d^{n+\frac{1}{2}}$. According to [4], using Taylor expansion we have

$$\Phi_d^{n+\frac{1}{2}} = \Phi^n + \frac{\Delta}{2}\frac{\partial \Phi^n}{\partial x_d} + \frac{\Delta t}{2}\frac{\partial \Phi^n}{\partial t} + O(\Delta^2, \Delta t^2) \tag{5}$$

Replacing $\frac{\partial \Phi^n}{\partial x_d}$ with equation 1 yielding

$$\Phi_d^{n+\frac{1}{2}} = \Phi^n + [\frac{\Delta}{2} - \frac{\Delta t}{2}\mathbf{u}^n \cdot \mathbf{n}_d]\frac{\partial \Phi^n}{\partial x_d} - \frac{\Delta t}{2}\mathbf{u}^n \cdot \mathbf{n}_e \frac{\partial \Phi^n}{\partial x_e} - \frac{\Delta t}{2}\mathbf{g}^n \tag{6}$$

Subscript e represents direction other than current direction d. Compromise is taken for sake of convenience (maybe) that $\mathbf{u}^n$ in equation 6 is replaced by $\mathbf{u}^{n+\frac{1}{2}}$ so that $\mathbf{u}^n \cdot \mathbf{n}_d$ can be computed by $u_d^{n+\frac{1}{2}}$. (see Sec.2.1.2 for more details).

2 Functions

2.1 tracer_fluxes

2.1.1 Parameters

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

2.1.2 Worth Mentioning Details

Data of *face vector* type is stored on staggered mesh[2]. Take *uf* as an example, the storage on a 2D cell displays in Fig.1.

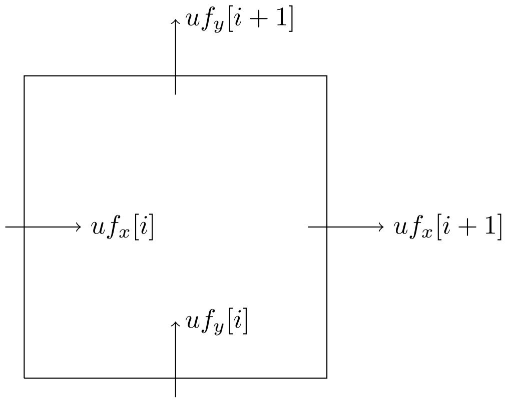

Figure 1: Staggered mesh.

Components are saved in corresponding face with single value. The direction of the component express itself in the feature of the cell face. Therefore, $\mathbf{u}^{n+\frac{1}{2}} \cdot \mathbf{n}_x$ can simply be $\frac{1}{2}(uf_x[i] + uf_x[i+1])$.

Advection term employs BCG scheme[3], a second order upwinded scheme. It first requires face-centered approximation of velocity $u_d^n, v_e^n(w_f^n)$. Then equation 6 is adapted to a upwinded scheme

$$\Phi_d^{n+\frac{1}{2}} = \begin{cases} \bar{\Phi}_d^{L,n+\frac{1}{2}} & if\ u_d^n > 0, \\ \bar{\Phi}_d^{R,n+\frac{1}{2}} & if\ u_d^n < 0, \\ \frac{1}{2}(\bar{\Phi}_d^{L,n+\frac{1}{2}} + \bar{\Phi}_d^{R,n+\frac{1}{2}}) & if\ u_d^n = 0. \end{cases} \tag{7}$$

where

$$\bar{\Phi}_d^{L,n+\frac{1}{2}} = \Phi^n[i] + \frac{\Delta}{2} min[1 - \frac{\Delta t}{2\Delta}(u_d^n[i] + u_d^n[i+1]), 1]\frac{\partial \Phi}{\partial x_d}[i] - \frac{\Delta t}{2}\mathbf{g}^n - flux_e[i] \tag{8}$$

$$\bar{\Phi}_d^{R,n+\frac{1}{2}} = \Phi^n[i+1] - \frac{\Delta}{2} min[1 - \frac{\Delta t}{2\Delta}(u_d^n[i] + u_d^n[i+1]), 1]\frac{\partial \Phi}{\partial x_d}[i+1] - \frac{\Delta t}{2}\mathbf{g}^n - flux_e[i+1] \tag{9}$$

$flux_e$ represents contribution from direction other than d, which also takes upwinded scheme

$$flux_e = \frac{\Delta t}{2}\mathbf{u}^n \cdot \mathbf{n}_e \frac{\partial \Phi^n}{\partial x_e} = \begin{cases} \frac{\Delta t}{2\Delta} v_e^{trans}(u^n[i,j] - u^n[i,j-1]) & if\ v_e^{trans} > 0, \\ -\frac{\Delta t}{2\Delta} v_e^{trans}(u^n[i,j] - u^n[i,j+1]) & if\ v_e^{trans} < 0. \end{cases} \tag{10}$$

where $v_e^{trans} = \frac{1}{2}(v_e^{trans}[i,j] + v_e^{trans}[i,j+1])$

2.1.3 Program Workflow

Starting Point

input:

$f = \Phi^n$ $uf = u_d^{n+\frac{1}{2}}$

$flux$(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});
```

$\downarrow$

Main Direction

Traversal all directions, suppose current direction is d

compute un[a]:

un=$\frac{\Delta t}{\Delta}\mathbf{u}^{n+\frac{1}{2}} \cdot \mathbf{n}_{dp}$

$=\frac{\Delta t}{2\Delta}(uf_d[i] + uf_d[i + 1])/(fm.d[i] + fm.d[i + 1])$

$\approx \frac{\Delta t}{\Delta}\frac{uf_x[i]}{fm.x[i]}$

component of main direction[b]:

$f2 = \Phi^n + \frac{\Delta}{2} upwind[1 - \frac{\Delta t}{\Delta}\mathbf{u}^{n+\frac{1}{2}} \cdot \mathbf{n}_{dp}, -1 - \frac{\Delta t}{\Delta}\mathbf{u}^{n+\frac{1}{2}} \cdot \mathbf{n}_{dp}]\frac{\partial\Phi^n}{\partial x_d} + \frac{\Delta t}{2}\mathbf{g}_d^n$

[a]Subscript $*p$ refers to positive normal unit whose direction agrees with coordinate positive direction. See 2.2.2 for more information.

[b]See 2.1.2 for details about upwinded

```
  foreach_face() {
    double un = dt*uf.x[]/(fm.x[]*Delta +
    ↪   SEPS), s = sign(un);
    int i = -(s + 1.)/2.;
    double f2 = f[i] + (src[] +
    ↪   src[-1])*dt/4. + s*(1. -
    ↪   s*un)*g.x[i]*Delta/2.;
```

$\downarrow$

Other Directions

Traversal the rest directions, suppose current direction is $e, e \neq d$

compute vn,wn:

$vn(wn)$=$\mathbf{u}^{n+\frac{1}{2}} \cdot \mathbf{n}_{ep}$

$=(uf_e[i] + uf_e[i + 1])/(fm.e[i] + fm.e[i + 1])$

component of other direction:

$fyy(fzz)$= $\frac{\partial\Phi^n}{\partial x_e} \cdot \Delta$

$f2$ -= $\frac{\Delta t}{2}\mathbf{u}^n \cdot \mathbf{n}_{ep}\frac{\partial\Phi^n}{\partial x_e}$

```
    #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$=u_d^{n+\frac{1}{2}}\Phi_d^{n+\frac{1}{2}}$

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

2.2 advection

2.2.1 Parameters

Name	Data type	Status	Option	Representation (before/after)
tracers	scalar*	update	compulsory	Φ^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.2 as an example.

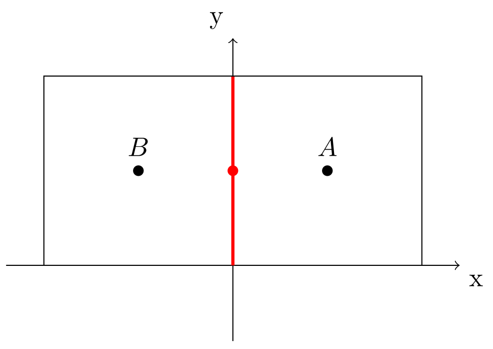

Figure 2: Face normal example.

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

$$u_d^{n+\frac{1}{2}}\Phi_d^{n+\frac{1}{2}}(i) + u_d^{n+\frac{1}{2}}\Phi_d^{n+\frac{1}{2}}(i+1) = -flux.x[i] + flux.x[i+1] \tag{11}$$

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));
```

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);
```

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 1, 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 6 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 1 is the cell-centered vector $\mathbf{u}^n$. According to Martin[3], normal components calculated by equation 6 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[4].

References

[1] John B Bell, Phillip Colella, and Harland M Glaz. "A second-order projection method for the incompressible Navier-Stokes equations". In: *Journal of computational physics* 85.2 (1989), pp. 257–283.

[2] Francis H Harlow and J Eddie Welch. "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface". In: *The physics of fluids* 8.12 (1965), pp. 2182–2189.

[3] Daniel F Martin and Phillip Colella. "A cell-centered adaptive projection method for the incompressible Euler equations". In: *Journal of computational Physics* 163.2 (2000), pp. 271–312.

[4] Stéphane Popinet. "Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries". In: *Journal of computational physics* 190.2 (2003), pp. 572–600.