Tracer Advection using Characteristic Discontinuous Galerkin

Robert B. Lowrie

October 31, 2012

Chapter 1

Summary

This document outlines the implementation of the Characteristic Discontinuous Galerkin (CDG) tracer advection scheme within the MPAS framework. In this document, we refer to the existing advection scheme as FVM, for finite-volume method.

1.1 TODO

- Add limiter algorithm. This will probably change the overall algorithm quite a bit, but I still want the base algorithm documented as such, since there may be many other ways to limit than the current approach.

Chapter 2

Requirements

1. Horizontal and vertical advection of tracers. The horizontal and vertical updates may be spatially-split, as they are in the existing advection module. Comments:
   (a) We might want to use CDG in the horizontal and FVM in the vertical, and vice versa. However, how does FVM update the CDG polynomial coefficients, aside from the cell-average? Consequently, for now, the same method must be used for both horizontal and vertical.
2. Preserves physical bounds on tracer quantities. This may also be extended to monotone advection, a more stringent requirement.
3. User-selectable polynomial degree for the representation of the tracers. For example, bi-quadratic would be analogous to Prather's method. Comments:
   (a) At this point, the polynomial degree will be the same across all mesh cells and for all tracers.
   (b) We likely need "diamond truncation" (linear example: $\{1, x, y, xy\}$) versus "triangle truncation" ($\{1, x, y\}$), but we might want to make this an option.
4. Operates within current FVM implementation.
5. The height-function update is performed elsewhere by MPAS. A constant tracer field must maintain this update, to within round-off.
6. Threadable and bit reproducible.

7. Allocate additional tracer polynomial coefficients only if CDG is selected as the advection scheme. The tracer cell-average will be CDG's leading coefficient. Comments:

   (a) Might just add another index to `tracers` array, with the first index the cell average. Another possibility is to store separately the CDG tracers and FVM tracers.

8. In order to share this module with other dynamical cores, isolate code dependencies on MPAS.

9. Isolate polygon intersections for remap step. Start with CORE, and if available and useful via SciDAC3, consider using MOAB.

10. The time step is limited by the number of neighboring cells one intersects with each face's Lagrangian pre-image. The algorithm should allow this "halo" (or "depth") to be variable and likely will use whatever halo is used by the current code, with its implied time step limitation.

Chapter 3

Formulation for Horizontal Advection

3.1 Equations Solved

The tracer advection system can be written as follows:

$$\partial_t h + \nabla \cdot (h\vec{u}) = 0\,, \tag{3.1a}$$

$$\partial_t (hq) + \nabla \cdot (hq\vec{u}) = 0\,, \tag{3.1b}$$

where $h(\vec{x}, t)$ is the height field and $q(\vec{x}, t)$ the tracer.

At each time-level t^n, in each cell Ω_z, we represent the tracer as

$$q(\vec{x}, t^n) = \sum_{j=1}^{N} c_{z,j}^n \beta_{z,j}(\vec{x})\,, \quad \vec{x} \in \Omega_z\,, \tag{3.2}$$

where the coefficients $\{c_{z,j}^n\}_{j=1}^N$ are to be updated by the CDG method. The choice of basis functions $\{\beta_{z,j}(\vec{x})\}_{j=1}^N$ will be discussed in §3.6.

To update the polynomial coefficients, we solve the system

$$h_z^{n+1} \int_{\Omega_z} \beta_{z,i} q^{n+1}\, d\Omega - h_z^n \int_{\Omega_z} (\phi_{z,i} q)^n\, d\Omega + \sum_{f \in \mathcal{F}(z)} \frac{m_f^n}{V_f} \int_{\Delta\Omega_f} (\phi_{z,i} q)^n\, d\Omega \;= 0\,, \tag{3.3}$$

for $i = 1, \ldots, N$, where m_f^n is the height flux through face-f (with the proper sign with respect to cell-z), $\Delta\Omega_f$ is the Lagrangian pre-image of face-f, $V_f = |\Delta\Omega_f|$, and $\mathcal{F}(z)$ is the set of faces that make up cell-z. The function $\phi_{z,i}(\vec{x}, t^n)$ will be discussed in §3.2.

Substitute (3.2) into the first integral in (3.3) to obtain

$$h_z^{n+1} M_{z,i,j} c_{z,j}^{n+1} - h_z^n \int_{\Omega_z} (\phi_{z,i} q)^n \, d\Omega + \sum_{f \in \mathcal{F}(z)} \frac{m_f^n}{V_f} \int_{\Delta\Omega_f} (\phi_{z,i} q)^n \, d\Omega \; = 0 \,, \quad (3.4)$$

where for each cell-z, $M_{z,i,j}$ is an $N \times N$ matrix, given by

$$M_{z,i,j} = \int_{\Omega_z} \beta_{z,i} \beta_{z,j} \, d\Omega \,. \quad (3.5)$$

The remaining integrals are a function of time-level-n data, so that (3.4) shows the manner in which the coefficients $c_{z,j}^{n+1}$ may be computed. For each time step, in order to expedite the computation of $c_{z,j}^{n+1}$, the matrix inverse $M_{z,i,j}^{-1}$ may be computed once and stored. More discussion of this issue will be given in §3.6. Indeed, the remainder of this chapter discusses how the terms in (3.4) are computed.

3.2 Determination of $\phi_{z,i}(\vec{x}, t^n)$

Within each time slab $t^n \le t \le t^{n+1}$, the function $\phi_{z,i}(\vec{x}, t)$ is determined by integrating

$$\frac{D\phi_{z,i}}{Dt} = 0 \,, \quad (3.6)$$

with $\phi_{z,i}(\vec{x}, t^{n+1}) = \beta_{z,i}(\vec{x})$. An alternative form is

$$\phi_{z,i}(\vec{x}, t) = \beta_{z,i}(\vec{\Gamma}(\vec{x}, t)) \,, \quad (3.7)$$

where

$$\vec{\Gamma}(\vec{x}, t) = \vec{x} + \int_t^{t^{n+1}} \vec{u}(\vec{\Gamma}(\vec{x}, \xi), \xi) \, d\xi \,. \quad (3.8)$$

For a constant velocity field, we have that

$$\phi_{z,i}(\vec{x}, t^n) = \beta_{z,i}(\vec{x} + \vec{u}\Delta t) \,. \quad (3.9)$$

In general, (3.8) may be integrated using Runge-Kutta, or some other time-integration scheme, to determine $\phi_{z,i}(\vec{x}, t^n)$. See §3.4 for more discussion.

3.3 Remap

This section describes the formation of the Lagrangian pre-image, $\Delta\Omega_f$, which can be viewed as a "remap" step. For each face-f, we approximate the pre-image and compute the integral as follows:

1. Find the departure points $\vec{d_1}$ and $\vec{d_2}$ for face endpoints $\vec{x}_1$ and $\vec{x}_2$. In general, a departure point is found via a relation very similar to (3.8). See §3.4 for more information.

2. Determine whether the line $(\vec{d_1}, \vec{d_2})$ intersects $(\vec{x}_1, \vec{x}_2)$.

   - If the lines do not intersect, then $\Delta\Omega_f$ is the quadrilateral with points $\{\vec{x}_1, \vec{x}_2, \vec{d_2}, \vec{d_1}\}$.
   - If the lines intersect, let $\vec{x}_I$ be the intersection point. Then $\Delta\Omega_f$ is made up of two triangles, $\{\vec{x}_1, \vec{x}_I, \vec{d_1}\}$ and $\{\vec{x}_2, \vec{x}_I, \vec{d_2}\}$.

3. Intersect $\Delta\Omega_f$ with any number of cells that neighbor the face. The larger the number of cells considered, the larger the maximum allowable Δt. See Requirement (10).

4. For each region found in the previous step, subdivide into triangles and integrate numerically. The details of this integration will be given in §3.7.

We note that the majority of this algorithm can be performed outside of the inner-loop over tracers.

3.4 Characteristic Tracing

The velocity characteristics must be traced in two places for the update:

1. To determine $\phi(\vec{x}, t^n)$ in the second and third integrals of eq. (3.3). Specifically, given a Gauss point $\vec{x}_g$ at $t = t^n$, we must integrate (3.8) to find where it lands at time $t = t^{n+1}$.

2. To determine the departure points $(\vec{d_1}, \vec{d_2})$, for each of the face's endpoints $(\vec{x}_1, \vec{x}_2)$, as described in §3.3. Specifically, given $(\vec{x}_1, \vec{x}_2)$ at time $t = t^{n+1}$, their departure points are found at $t = t^n$.

The difficulty is that the velocity field is discrete. In this section, we describe a second-order accurate, predictor-corrector algorithm. For Case 1, we have the steps:

1. Interpolate edge velocities $\vec{u}_e^n$ to at $\vec{x}_g^n$ to find $\vec{u}_I^n$.
2. Predict: $\vec{x}_g^{n+1} = \vec{x}_g^n + \Delta t \vec{u}_I^n$
3. Interpolate edge velocities $\vec{u}_e^{n+1}$ at $\vec{x}_g^{n+1}$ to find $\vec{u}_I^{n+1}$
4. Let $\vec{u}_I^{n+1/2} = (\vec{u}_I^n + \vec{u}_I^{n+1})/2$
5. Correct: $\vec{x}_g^{n+1} = \vec{x}_g^n + \Delta t \vec{u}_I^{n+1/2}$

Assuming the velocity interpolation is at least second-order accurate in space, this algorithm is second-order accurate in space and time.

Case 2 is very similar. For a given vertex $\vec{x}_i$:

1. Interpolate edge velocities $\vec{u}_e^{n+1}$ to $\vec{x}_i$ to find $\vec{u}_I^{n+1}$.
2. Predict: $\vec{d_i} = \vec{x}_i - \Delta t \vec{u}_I^{n+1}$
3. Interpolate edge velocities $\vec{u}_e^n$ at $\vec{d_i}$ to find $\vec{u}_I^n$
4. Let $\vec{u}_I^{n+1/2} = (\vec{u}_I^n + \vec{u}_I^{n+1})/2$
5. Correct: $\vec{d_i} = \vec{x}_i - \Delta t \vec{u}_I^{n+1/2}$.

3.5 Mass Conservation and the Preservation of a Constant Tracer

Any basis will be a "partition of unity." Specifically, for each cell-z, there exists constants $\{\alpha_j\}_{j=1}^N$ such that

$$\sum_{i=1}^{N} \alpha_i \beta_{z,i}(\vec{x}) = 1\,, \tag{3.10}$$

for all $\vec{x}$. As a result, if we take this same linear combination of (3.3), and assume that q is a constant, then (3.3) reduces to

$$V_z(h_z^{n+1} - h_z^n) + \sum_{f \in \mathcal{F}(z)} m_f^n = 0\,. \tag{3.11}$$

This is simply the mass balance over the cell, corresponding to (3.1a). We assume that the flow solver satisfies (3.11), and this is the sense that Requirement (5) is satisfied. Note that for MPAS,

$$m_f^n = \frac{1}{2}\Delta t (h_z^n + h_{z'}^n)\vec{u}_f^n \cdot \vec{n}_f\,, \tag{3.12}$$

where cell-z' is the cell neighbor across face-f, and the area-weighted normal $\vec{n}_f$ points in this direction.

3.6 Choice of Basis Functions

This section discusses the choice of basis functions $\{\beta_{z,j}(\vec{x})\}_{j=1}^{N}$ in eq. (3.2). Many DG implementations use an orthonormal basis on each cell, but such a basis does not exist for hexagonal cells. Two options are discussed in detail in this section.

To begin with, the basis functions may be selected independent of the cell-index k as the set of polynomials

$$\beta_{z,j} = \mathcal{P}_j = x^{(j-1) \bmod (p+1)} y^{\lfloor (j-1)/(p+1) \rfloor} . \tag{3.13}$$

where p is the polynomial order and $1 \leq j \leq (p+1)^2$. For $p = 2$ (bi-quadratic), $N = 9$ in (3.2), and we have

$$\{\mathcal{P}_j\}_{j=1}^{N} = \{1, x, x^2, y, xy, x^2y, y^2, xy^2, x^2y^2\} . \tag{3.14}$$

This choice of basis is not ideal, for a number of reasons, all of which we won't go into here.

One issue is that a straightforward way to meet Requirement (7) is for

$$c_{z,1}^n = \bar{q}_z^n \equiv \frac{1}{V_z} \int_{\Omega_z} q(\vec{x}, t^n) d\Omega , \tag{3.15}$$

where $\bar{q}_z^n$ is the cell average. The basis (3.13) can be adjusted to satisfy this property with

$$\beta_{z,j} = \mathcal{P}_j - b_{z,j} , \tag{3.16}$$

where for each cell, the constants $\{b_{z,j}\}_{j=1}^{N}$ are given by

$$b_{z,j} = \begin{cases} 0 & \text{for } j = 1, \\ \int_{\Omega_z} \mathcal{P}_j d\Omega & \text{otherwise.} \end{cases} \tag{3.17}$$

This can viewed as the first step of a Gram-Schmidt orthogonalization.

Indeed, a Gram-Schmidt orthogonalization of $\mathcal{P}_j$ is given by the recursion relation

$$\beta_{z,j} = \mathcal{P}_j - \sum_{k=1}^{j-1} g_{z,j,k} \beta_{z,k} \tag{3.18}$$

where

$$g_{z,j,k} = \int_{\Omega_z} \mathcal{P}_j \, \beta_{z,k} d\Omega \Big/ \int_{\Omega_z} \beta_{z,k}^2 d\Omega \ . \tag{3.19}$$

We claim that we can then write (3.18) as

$$\beta_{z,j} = \sum_{k=1}^{j} L_{z,j,k} \mathcal{P}_k \tag{3.20}$$

where for each cell-z, $L_{z,j,k}$ is an $N \times N$, lower-triangular matrix. The precise form of $L_{z,j,k}$ won't be given here, but we claim it may be computed so that

$$M_{z,i,j} = V_z \delta_{i,j} \, , \tag{3.21}$$

which might appear to greatly decrease the computation cost for (3.4).

However, consider the costs of using the non-orthogonal basis (3.16) versus the orthonormal basis (3.20). Assume the update (3.3) is performed inside a cell loop, so that Requirement (6) is satisfied. Then the costs may be summarized as follows:

- **Use of non-orthogonal basis (3.16)**: For each cell-z, the following must be computed:
  - The matrix $M_{z,j,k}$, which has N^2 elements.
  - The inverse $M_{z,j,k}^{-1}$. This computation costs approximately $O(N^3)$ FLOPS. Note that $M_{z,j,k}^{-1}$ may be pre-computed and stored, outside of the cell-update loop, and then $M_{z,j,k}$ discarded.
  - The array $b_{z,j}$, which contains N elements.
- **Use of orthonormal basis (3.20)**:
  - For each cell-z, the matrix $L_{z,j,k}$ must be computed, which has $N^2/2$ nonzero elements.
  - For each quadrature point used for the integrals in (3.4), the matrix-vector product (3.20) requires $O(N^2)$ extra FLOPS, compared with using the non-orthogonal basis. There is an absolute minimum of N^2 quadrature points (and typically many more), so that at least $O(N^4)$ extra FLOPS are required.
  - If $L_{z,j,k}$ is computed within the cell-update loop, rather than pre-computed, then it must also be computed for all of cell-z's neighbors that are covered by a Lagrangian pre-image. In contrast, $M_{z,j,k}^{-1}$ needs to be computed only for cell being updated.

Note that such "on-the-fly" computation may be used for certain computer architectures that have low-memory bandwidth.

Consequently, at this time, we will use the non-orthogonal basis (3.16).

3.7 Quadrature

For the mesh cells used in MPAS, all of the integrals in (3.3) are computed with quadrature. Generally, the integration regions are triangulated, and each triangle is integrated with known integration weights and points. For the region Ω_z, the triangulation is formed by computing the centroid and then connecting the centroid with each of the cell's vertices. For N^{vertices} vertices, this forms $N^{\text{tri}}(\Omega_z) = N^{\text{vertices}} - 1$ triangles.

On each triangle, we use the same quadrature rule with N^{gauss} points, so that for example the second integral in (3.3), we have

$$\int_{\Omega_z} (\phi_{z,i} q)^n \, d\Omega \approx \sum_{it=1}^{N^{\text{tri}}(\Omega_z)} A_{it} \sum_{g=1}^{N^{\text{gauss}}} w_g (\phi_{z,i}^n q^n)|_{\vec{x}=\vec{x}_g} \,, \tag{3.22}$$

where A_{it} is the triangle's area and $\vec{x}_g$ is the Gauss-point location, with corresponding integration weight w_g. We assume here that the weights are normalized such that

$$\sum_{g=1}^{N^{\text{gauss}}} w_g = 1 \,, \tag{3.23}$$

and that the A_{it} satisfy

$$V_z = \sum_{it=1}^{N^{\text{tri}}(\Omega_z)} A_{it} \,. \tag{3.24}$$

As part of a pre-processing step, we can accumulate all the Gauss points and weights into a single array for cell-z, so that (3.22) takes the form

$$\int_{\Omega_z} (\phi_{z,i} q)^n \, d\Omega \approx V_z \sum_{g=1}^{N_z^{\text{gauss}}} w_{z,g} (\phi_{z,i}^n q^n)|_{\vec{x}=\vec{x}_{z,g}} \,, \tag{3.25}$$

where

$$N_z^{\text{gauss}} = N^{\text{gauss}} N^{\text{tri}}(\Omega_z) \,, \tag{3.26}$$

$$w_{z,g'} = w_g A_{it} / V_z \,, \tag{3.27}$$

with $g' = g + (z-1)N^{\text{gauss}}$.

We strive to perform as much computation as possible, outside of the loop over tracers. To this end, we use (3.2) and write (3.25) as

$$\int_{\Omega_z} (\phi_{z,i} q)^n \, d\Omega \approx V_z \sum_{j=1}^{N} c_{z,j}^n \sum_{g=1}^{N_z^{\text{gauss}}} w_{z,g} \phi_{z,i}^n(\vec{x}_{z,g}) \beta_{z,j}(\vec{x}_{z,g}) \,. \tag{3.28}$$

As an initialization step, before the time-step loop, we can compute

$$B_{z,j,g} = V_z w_{z,g} \beta_{z,j}(\vec{x}_{z,g}) \,, \tag{3.29}$$

which requires storing a $N \times N_z^{\text{gauss}}$ matrix for each cell-z. Within the time-step loop, we can evaluate ϕ for all tracers, and compute

$$K_{z,i,j}^n = \sum_{g=1}^{N_z^{\text{gauss}}} B_{z,j,g} \phi_{z,i}^n(\vec{x}_{z,g}) \,, \tag{3.30}$$

so that (3.28) reduces to

$$\int_{\Omega_z} (\phi_{z,i} q)^n \, d\Omega \approx \sum_{j=1}^{N} K_{z,i,j}^n c_{z,j}^n \,; \tag{3.31}$$

that is, a matrix-vector product. Note that if the overall update is within a cell loop, then $K_{z,i,j}^n$ can be computed within this loop, and there is no need to permanently store it for each cell-z.

The integration over each Lagrangian pre-image may be computed in a similar fashion. Because the velocity is time dependent, the pre-image region is also time dependent, so all of the setup must be within the time-step loop. However, much of the work may be completed before the inner tracer loop. To begin with, the integration over the pre-image may be written as:

$$\int_{\Delta\Omega_f} (\phi_{z,i} q)^n \, d\Omega \;\; = \sum_{z'} \int_{\omega_{z'}} (\phi_{z,i} q)^n \, d\Omega \tag{3.32}$$

where z' sums over the halo of neighbors of face-f and $\omega_{z'} = \Omega_{z'} \cap \Delta\Omega_f$. Note here that ϕ is with respect to the basis in cell-z, rather than cell-z'. As discussed in §3.3, $\Delta\Omega_f$ may itself consist of two separate triangles ("Case 2"), but this case is an obvious extension of the case when $\Delta\Omega_f$ is a single

region ("Case 1"). Next, we write the integral above in a similar form as (3.31):

$$\int_{\omega_{z'}} (\phi_{z,i} q)^n \, d\Omega \;\approx \sum_{j=1}^{N} F^n_{z',i,j} c^n_{z',j} \tag{3.33}$$

For each cell-neighbor z', the $N \times N$ matrix $F^n_{z',i,j}$ is computed by subdividing $\omega_{z'}$ into triangles and using quadrature on each of these triangles:

$$F^n_{z',i,j} = \sum_{it=1}^{N^{\text{tri}}(\omega_{z'})} A_{it} \sum_{g=1}^{N^{\text{gauss}}} w_g \phi^n_{z,i}(\vec{x}_g) \beta_{z',j}(\vec{x}_g) \,. \tag{3.34}$$

However, eqs. (3.33-3.34) get only the contribution of z' for a single face-f. Referring to (3.3), we can sum the contributions of all faces as

$$\sum_{f \in \mathcal{F}(z)} \frac{m^n_f}{V_f} \int_{\Delta\Omega_f} (\phi_{z,i} q)^n \, d\Omega \approx \sum_{z'} \sum_{j=1}^{N} F^n_{z',i,j} c^n_{z',j} \tag{3.35}$$

where now z' sums over union of all the face's neighbors and

$$F^n_{z',i,j} = \sum_{f \in \mathcal{F}(z)} \frac{m^n_f}{V_f} \sum_{it=1}^{N^{\text{tri}}(\omega_{z'})} A_{it} \sum_{g=1}^{N^{\text{gauss}}} w_g \phi^n_{z,i}(\vec{x}_g) \beta_{z',j}(\vec{x}_g) \,. \tag{3.36}$$

3.8 Initial Condition

Given a tracer initial condition $q^0(\vec{x})$, its initial expansion coefficients are computed with the matrix-vector product

$$c^0_{z,i} = M^{-1}_{i,j} v^0_{z,j} \,, \tag{3.37}$$

where

$$v^0_{z,j} = \int_{\Omega_z} q^0 \beta_{z,j} \, d\Omega \,. \tag{3.38}$$

This integral is computed with the same quadrature rule as in (3.25).

3.9 Pseudocode Formulation

The overall update is broken into two main loops over the cells:

1. *Initial setup*: Initial computations that are performed before entering the time-advance loop. These computations are given in Algorithm 1.

2. *Update loop*: This loop is summarized in Algorithm 2, with the details given in Algorithms 3 and 4. This algorithm is within the time-step loop. Note that there are many computations that are common to all tracers (Algorithm 3), performed before a loop over all tracers (Algorithm 4).

Note that if one does not want to store quantities performed in the initial setup, such as the matrix $M^{-1}_{z,i,j}$ for all cells, then the two cell loops of Algorithms 1-2 may be combined.

Algorithm 1 Initial setup. We may place this loop outside of the time-step loop.

Load Gauss points and weights for triangle
for $z = 1, N^{\text{cells}}$ **do** ▷ loop over all owned cells
  Compute centroid of cell-z
  Compute triangulation of each cell-z
  $N^{\text{tri}}(\Omega_z) = N_z^{\text{vertices}} - 1$
  for $it = 1, N^{\text{tri}}(\Omega_z)$ **do** ▷ loop over triangles for this cell
    Compute area of triangle, A_{it}
    Normalize triangle Gauss weights by A_{it}/V_z
    Accumulate new weights and points into $w_{z,g}$ and $\vec{x}_{z,g}$
  end for
  Compute $M^{-1}_{z,i,j}$ ▷ using quadrature set above
  Compute $b_{z,j}$ ▷ using quadrature set above
  Compute $B_{z,j,g}$ ▷ see eq. (3.29)
  Compute $v^0_{z,j}$ ▷ using quadrature set above
  Initialize $c^0_{z,i}$ ▷ see eq. (3.37)
end for
return $M^{-1}_{z,i,j}$, $b_{z,j}$, $B_{z,j,g}$, $c^0_{z,i}$

Algorithm 2 Main update loop.

Fill off-processor cell neighbors with data
for $z = 1, N^{\text{cells}}$ **do** ▷ loop over all owned cells
    Computations shared by all tracers, Algorithm 3
    Update tracers, Algorithm 4
end for ▷ cell loop

Algorithm 3 Computations shared by all tracers, within main update loop (see Algorithm 2). Even though $K_{z,i,j}$ has a cell-index z, it does not need to be stored for all cells. For $F_{z',i,j}$, z' spans the cells that intersect the Lagrangian pre-image of any face-f.

$K^n_{z,i,j} = 0$ ▷ initialize array
for $i = 1, N$ **do** ▷ loop over number of basis
    for $g = 1, N_z^{\text{gauss}}$ **do** ▷ loop over gauss points for this cell
        Compute $\phi^n_{z,i}(\vec{x}_{z,g})$
        for $j = 1, N$ **do**
            $K^n_{z,i,j} = K^n_{z,i,j} + B_{z,j,q}\phi^n_{z,i}(\vec{x}_{z,g})$ ▷ see eq. (3.30)
        end for
    end for
end for
$F^n_{z',i,j} = 0$ ▷ initialize array
for $f = 1, N_z^{\text{faces}}$ **do** ▷ loop over faces for this cell
    Compute Lagrangian pre-image of face-f...
    ... to form subregions ω_p, $p = 1, N^{\text{preimages}}$ ▷ see §3.3
    for $zf = 1, N_f^{\text{neighbors}}$ **do** ▷ loop over face's cell neighbors
        for $p = 1, N^{\text{preimages}}$ **do** ▷ loop over preimage regions
            Intersect ω_p with Ω_{zf} to form R
            if R is not an empty region **then**
                Put zf in list of cell-z's neighbors, with linear index z'
                Split R into triangles
                Integrate over triangles and sum contributions...
                ... to $F^n_{z',i,j}$ and V_f ▷ see eq. (3.36)
            end if
        end for ▷ pre-image loop
    end for ▷ cell-neighbor loop
end for ▷ face loop
return $K^n_{z,i,j}$, $F^n_{z',i,j}$, list of cells that z' spans

Algorithm 4 Update tracers, within main update loop (see Algorithm 2)

1: **for** $iq = 1, N^{\text{tracers}}$ **do** ▷ loop over number of tracers
2: $sumFlux_i = 0$ ▷ N-array to accumulate fluxes
3: $sumFlux_i = sumFlux_i + K_{z,i,j} c^n_{z,j}$ ▷ matrix-vector multiply
4: **for** $z' = 1, N_z^{\text{neighbors}}$ **do** ▷ loop over cell's neighbors
5: $sumFlux_i = sumFlux_i - F_{z',i,j} c^n_{z',j}$ ▷ matrix-vector multiply
6: **end for**
7: $c^{n+1}_{z,iq,i} = M^{-1}_{z,i,j}(sumFlux_j)$ ▷ matrix-vector multiply
8: **end for** ▷ tracer loop

Chapter 4

Formulation for Vertical Advection

To be written.

Chapter 5

Design and Implementation

5.1 Namelist Options

The namelist options are as follows:

- `config_tracer_adv_meth`
  This is a flag corresponding to either the CDG or FVM methods. The options for FVM are as before. The options for CDG are given below.
  Available options: CDG or FVM

- `config_cdg_ord`
  Polynomial order of basis, for all cells.
  Available options: Any non-negative integer, although effectively limited by `config_cdg_gauss`

- `config_cdg_gauss`
  Max polynomial order integrated exactly by quadrature on triangles. If -1, estimate from `config_cdg_ord`. Note that general quadrature on triangles, for any order, does not exist.
  Available options: 0 to 25

- `config_cdg_limit`
  Selects how tracer bounds are computed by the limiter.
  `none` means don't limit at all.
  `local` means use local cell averages to compute the bounds.
  `global` means use pre-computed, global minimum and maximums.
  Available options: none, bounds, global

5.2 Implementation Issues

- Use `advCellsForEdge` index array as the cell halo to search for each edge's Lagrangian pre-image.
- §3.4 requires the edge velocities to be interpolated to any spatial coordinate, at either $t = t^n$ or $t = t^{n+1}$. This interpolation call should be its own routine, so various methods may be explored.