Finite Volume Methods on Unstructured Meshes

Mech 511

Learning Objectives

As a result of this handout and in-class discussions, students will be able to:

- Define an unstructured mesh.
- Outline the solution process for a time-accurate problem on unstructured meshes.
- Derive expressions for first- and second-order accurate flux integral evaluations for the two-dimensional advection problem on a triangular mesh with cell-centered control volumes.
- Describe the general approach taken in reconstructing a function on an unstructured mesh.

1 Introduction

Structured mesh methods are very efficient and not too difficult to apply for most cases with relatively simple geometry. Alas, the real world does not always present us with simple problem geometries. As we discussed when talking about structured mesh generation, there are techniques for getting around this difficulty, but they are not trivial. Figure 1 show two pictures of an unstructured triangular mesh generated for the branched-duct geometry that we talked about earlier, in conjunction with structured mesh generation.

An alternate approach, which is becoming the dominant approach, especially in commercial CFD codes, is the use of unstructured meshes, which offer a great deal more flexibility geometrically than structured meshes. Why is this? Is it because unstructured meshes typically use triangular cells, while structured meshes use quadrilaterals? No. After all, the two-hole geometry of Figure 2a is an unstructured mesh

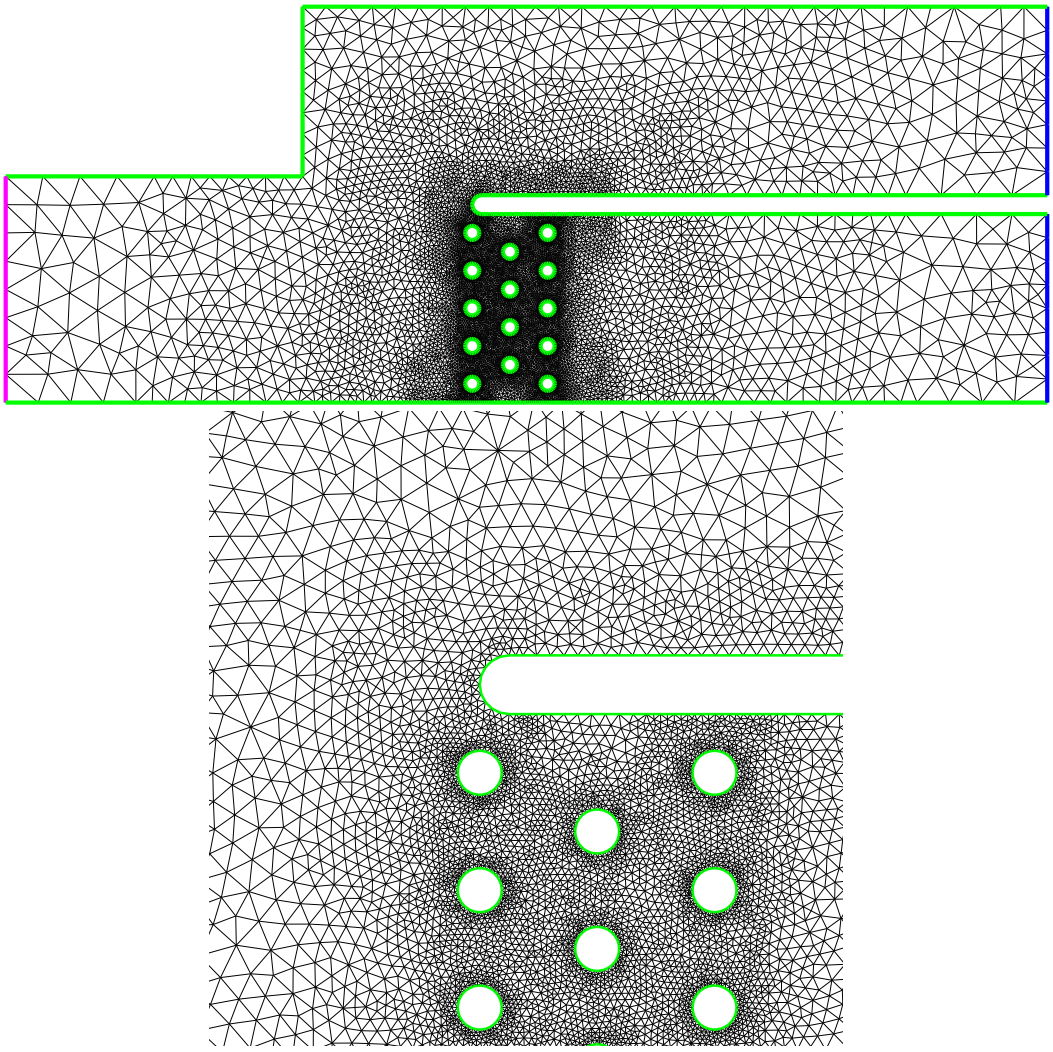
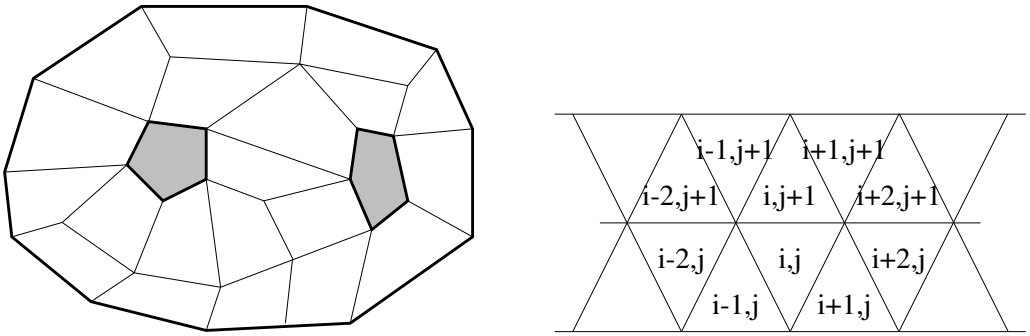


Figure 1: Branched-duct mesh, and close-up near splitter plate.



(a) Unstructured quadrilateral mesh.

(b) Structured triangular mesh.

Figure 2: Mesh classification examples

of quadrilaterals (not necessarily a good one in this case, but that isn't the point). On the other hand, the mesh fragment of Figure 2b is structured. So what is the distinction? I prefer the following definition.

Definition. Regardless of cell shape, in a *structured mesh,* the cells which neighbor a given cell can be determined solely by manipulating the indices of the given cell. For structured quadrilateral meshes, the neighbors of a given cell i, j are known to be $i \pm 1, j \pm 1$. For the structured triangular mesh of Figure 2b, the neighbors of cell i, j are $i \pm 1, j$ and either $i, j + 1$ or $i, j - 1$, depending on whether $i + j$ is even or odd.

Definition. Any other mesh is *unstructured*, regardless of what kind of cell shapes in contains or even if it contains more than one type of cell.

1.1 Control Volume Definitions

Despite the generality of the definitions given above, we will focus solely on unstructured meshes consisting entirely of triangles. There are two common ways of dividing such a mesh into control volumes.[1] Vertex-centered control volumes give one CV per vertex in the mesh by connecting edge midsides to cell centroids. Cell-centered control volumes given one CV per cell (triangle, quadrilateral, tetrahedron, whatever). These two alternatives are shown in Figure 3. We'll focus on cell-centered control volumes for geometric simplicity. Also, we will do no more than touch briefly on the issue of implicit schemes on unstructured meshes, because the numerical linear algebra difficulties there will cloud our view of more fundamental ideas.

[1] Actually, any unstructured mesh can be divided into control volumes using generalizations of either of these methods.

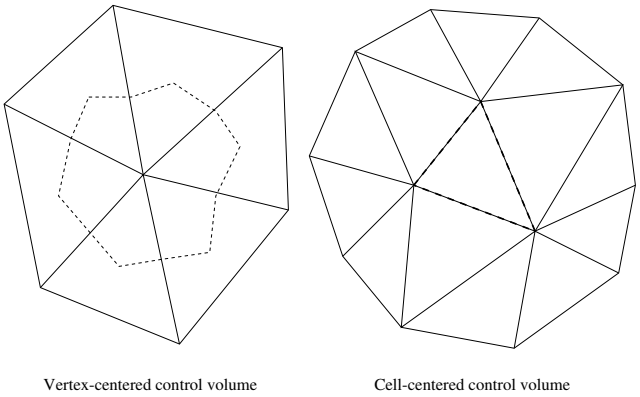


Figure 3: Control volumes for unstructured triangular meshes.

2 The Question of Connectivity

Not surprisingly, with this freedom in connectivity comes the responsibility of keeping track of connectivity information, which we didn't have to worry about with structured meshes. There are several options for connectivity storage, depending on which connectivity information needs to be available quickly and which information one can stand to calculate whenever it is needed. My group's research mesh generation code stores the following information:

- For each face ($d-1$ dimensional object), the identity of its vertices and the cells on either side of it.
- For each cell (d dimensional object), the identity of its bounding faces.
- For each vertex, the list of faces which contain it.
- For boundary faces, some additional information regarding boundary conditions and so forth.

This is more information than a flow solver typically needs to have readily available, although the information a flow solver requires varies tremendously depending on how the solver is constructed.

For a second-order finite-volume code using vertex-centered control volumes, the minimum connectivity data required by the solver at run-time is:

- For each edge, the identities and locations of its vertices.
- For each vertex, the identities of all neighboring vertices.

For a second-order finite-volume code using cell-centered control volumes, which is what we will discuss in what follows, the minimum connectivity data required at run-time is:

- For each cell, the locations of its vertices.
- For each cell, the identities of its neighboring cells (those with which it shares a face).
- For each face, the identities of its neighboring cells.

You can store less mesh data (bare minimum: for each vertex, its location; and for each cell, the identities of all its vertices), but if you do, you have to re-compute a lot of things on the fly.

3 Interlude

The rest of these notes will discuss techniques for unstructured mesh discretization by example, creating in the end all the pieces for a second-order accurate approximation to the two-dimensional advection-diffusion equation on a triangular mesh. That is, we will solve the following problem:

$$\frac{\partial a}{\partial t} + \frac{\partial au(x,y)}{\partial x} + \frac{\partial av(x,y)}{\partial y} = \kappa \left(\frac{\partial^2 a}{\partial y^2} + \frac{\partial^2 a}{\partial x^2} \right) \tag{1}$$

on the domain shown in Figure 4. Because this problem is elliptic in space, we need boundary conditions on a on all four boundaries. The boundary conditions are as follows:

- Curve D is an inflow boundary, where a is specified.
- Along curves A and C, the velocity perpendicular to the wall is zero, so there is no inviscid flux. We will enforce a Dirichlet condition on curve A and a Neumann condition on curve C.
- Along curve B, the outflow, we will take the boundary condition to be fully-developed flow. If $\kappa = 0$, then if we use upwind fluxes, we don't need a boundary condition.

Also, note that (u, v) is a given velocity vector field and is not a function of a.

I am convinced, and I expect to convince you, that you actually already know all but one of the basic concepts that are needed to solve this problem, although there are some other details that need consideration.

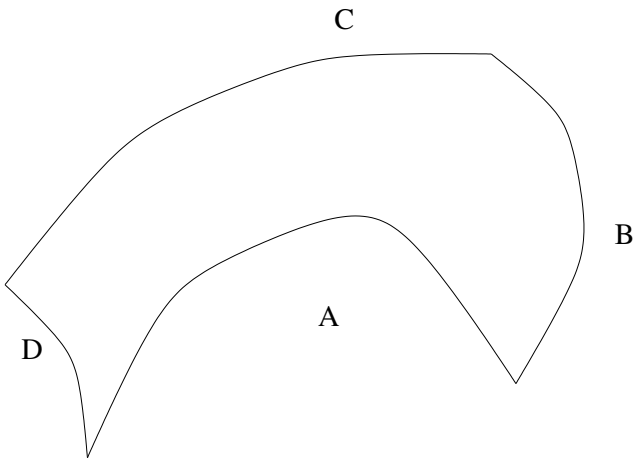


Figure 4: Geometry for two-dimensional advection problem.

4 Outline of a Generic Finite-Volume Unstructured Mesh Scheme

Development of an unstructured mesh finite-volume scheme is very similar in principle to a structured mesh scheme.[2] The procedure for a single time step of an explicit, unstructured mesh scheme will be:

1. Reconstruct the solution to the desired order of accuracy. This process allows us to replace the piecewise constant control volume average data in each control volume with a polynomial. We'll discuss the particular case of using a linear polynomial, which will give us a second-order accurate approximation to the solution. This is the one really new concept required for unstructured meshes versus structured meshes.

2. Use the reconstructed solution data, we can calculate the fluxes at whatever points we require it on the control volume boundaries.

3. For each face in the mesh, including boundary faces, compute the integral of the flux across that face. Note that we will often enforce boundary conditions using a special flux. This flux should be subtracted from the "donor" cell (the one out of which the face normal points) and added to the "receptor" cell (the one into which the face normal points). The result of this integration and

[2] As mentioned earlier, we will be discussing exclusively cell-centered control volumes, but the approach is exactly the same for vertex-centered control volumes. The main difference is that the geometric terms are evaluated differently.

summation will be the computation of the residual for each control volume. This residual can be written in the familiar mathematical form

$$R_i = \oint_{\partial CV_i} \vec{F} \cdot \vec{n}\, ds$$

4. While accumulating the residual, also accumulate the information needed to find the maximum stable time step for each control volume. The global maximum stable time step can be no larger than the minimum of these local values.

5. Advance the solution in time using your favorite explicit scheme. A first-order explicit time advance scheme can be written as

$$U_i^{n+1} = U_i^n + \Delta t\, R_i^n$$

where

$$\Delta t = \mathrm{CFL} \cdot \Delta t_{\max}$$

We will fill in the parts of this in the following order:

Section 5. Advective flux evaluation and integration, both in the interior and on the boundary, that produces a first-order accurate result.

Section 6. Determination of the largest stable explicit time step.

Section 7. Solution reconstruction.

Section 8. Advective and diffusive flux evaluation and integration, both interior and boundary, that produce a second-order accurate result.

While in principle these sections tell you everything you need to know to write an unstructured mesh flow solver, there is a lot of low-level data manipulation that is required to keep all the connectivity information straight. There are many ways of doing this of course, but Section 9 describes the approach used in my group's research code.

5 Advective Flux Evaluation and Integration (First Order Scheme)

For finite volume methods on structured meshes, we computed first-order upwind fluxes by determining the flow direction and simply using data from the upstream side. The same principle applies for unstructured mesh calculations. Once we have

evaluated the flux on a face, we also must perform a numerical integration along the face.

For the two-dimensional advection problem of Equation 1, we want to compute the first-order upwind flux on the interior face $\overline{ac}$ in Figure 5. We will assume that we know the coordinates of all the points, the velocity vector (u, v) at any (x, y), and the control volume average value of a for both CV i and CV j.

Clearly, there are two choices: the flow can be from CV 1 into CV 2, or the other way around. We can determine this by taking the dot product of the local velocity with a unit normal. So first we compute a unit normal, pointing from CV 1 into CV 2:

$$\hat{n}_{12} = \frac{1}{l_{ac}} \begin{pmatrix} y_c - y_a \\ -x_c + x_a \end{pmatrix}$$

where l_{ac} is the length of edge ac. The orientation of the unit normal doesn't have to be in any particular direction. That is, we don't require that the unit normals all point outward, because of the way we do the flux integration. Instead, we simply compute the unit normal so that it points to the right as you move along the edge from its "source" to "target" vertices. Then we add and subtract the flux integral for the edge from the total for each of its neighboring cells.

This is a difference from structured meshes, where we defined all unit normals to be outward. In that case we computed the flux integral for each control volume in one chunk. In the process, we duplicated the flux calculation for every face, which we aren't doing here. In practice, for problems with complicated fluxes, flux calculation is a significant enough part of total CPU time that we would need to modify what we did on structured meshes to reclaim half that CPU time.

Just as we did for structured meshes, we will use a single flux integration point and the midpoint of the edge. Therefore, the velocity that we need is the midpoint velocity:

$$\vec{u}_{ac} = \vec{u}\left(\frac{x_a + x_c}{2}, \frac{y_a + y_c}{2}\right)$$

If this dot product $\vec{u}_{ac} \cdot \hat{n}_{12}$ (that is, the normal component of velocity) is positive, then flow is from 1 into 2, and vice versa. In either case, we compute the flux as the normal velocity multiplied by the solution in the upwind cell:

$$F_{ac} = \begin{cases} \hat{n}_{12}\vec{u}_{ac}a_1 & \hat{n}_{12}\vec{u}_{ac} > 0 \\ \hat{n}_{12}\vec{u}_{ac}a_2 & \hat{n}_{12}\vec{u}_{ac} < 0 \end{cases}$$

Finally, the flux integral is just the flux times the edge length: $F_{ac}l_{ac}$. This is added to CV 2 (the one into which the normal points) and subtracted from CV 1 (the one from which the normal points).

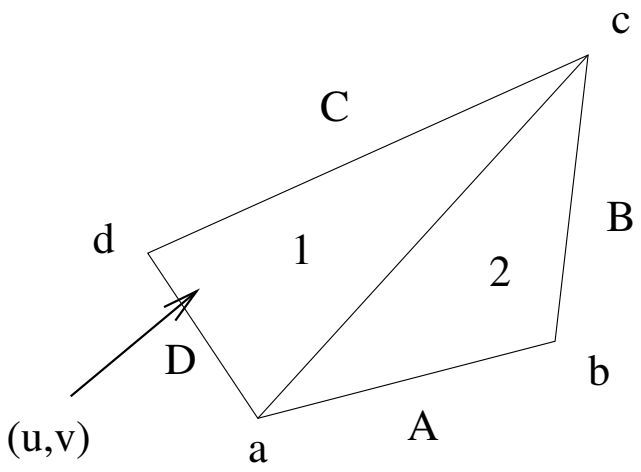


Figure 5: Control volumes for flux calculation

Of course, this isn't quite the whole story, as we need to be able to evaluate boundary fluxes as well. Suppose that boundary conditions are applied to the four outer faces as shown in Figure 5.[3] We know what the boundary fluxes *should* be from the mathematical boundary conditions. The question is, how do we evaluate them in practice?

First, on curves A and C, we know that there is no normal velocity, so there is no advective flux. Depending on the shape of our mesh boundary, we may or may not get exactly $\vec{u} \cdot \hat{n} = 0$ if we compute the normal velocity, so it's smarter to have a special wall boundary flux that gives exactly zero; that way, none of our solution quantity a will leak out (or in) through these sides.

At the inflow and outflow, we use the upwind scheme much as we did in the interior. For curve B, the flow is definitely out of the domain, so we always use data from CV 2; other than eliminating the check for the sign of the dot product, this is the same as the interior flux. For curve D, flow is into the domain. We again use the mid-edge velocity and the known boundary condition on a to compute this incoming flux.

Because the flux must be calculated across every face in the mesh while computing the residual, it only makes sense to use a single loop over all the faces to compute the residual. The normal to the face is needed to compute the flux, and should be computed using a right-hand rule. That is, if the interior face in Figure 5 is specified as $\overline{ac}$, then the normal should point into CV 2; otherwise, the normal should point into CV 1. With this convention, the flux (which may be negative if $\vec{u} \cdot \hat{n}$ is negative)

[3] Okay, so the mesh is a little too coarse. So what? For purposes of illustration, this is sufficient.

is added to the CV into which the normal points and subtracted from the other CV incident on the face. While this convention is not the only consistent choice, *some* consistent choice must be made, or the flux integrals will all be garbage.

Note that we can't really solve the full advection-diffusion problem with piecewise constant data, because we don't have a solution gradient. There are ways we could brew something up that would work [4], but instead we'll defer the diffusion terms until we have derivatives to work with.

6 Determining Maximum Stable Time Step

We're going to take a heuristic approach to determining the largest time step that's possible while maintaining stability. Note that it is possible to prove that the time step we derive is the correct one.

Consider the one-dimensional advection problem

$$\frac{\partial a}{\partial t} + c\frac{\partial a}{\partial x} = 0$$

For this problem with c constant, the maximum stable time step for explicit Euler time advance is

$$\Delta t_{\max} = \frac{\Delta x}{c} \tag{2}$$

You can easily verify this using von Neumann stability analysis or eigenvalue analysis. This result can be interpreted physically as saying that the maximum time step is limited so that no information can move more than a single cell in one time step. After all, information is moving at speed c, and the distance information travels in a single time step, $c\Delta t_{\max}$, is equal to the cell size Δx.

A subtly different way to look at this one-dimensional result — and a more useful one from our perspective — is that the time step is limited so that the flux into a cell in an upwind method (not the *net* flux, the *incoming* flux) can not be more than the amount that would completely fill the cell. That is, if the value at the left boundary of the cell is a_0 and the value *in* the cell is zero, with the maximum time step limit of Equation 2 the value in the cell at the next time step will not exceed a_0.

[4] Hint: *not* the directional derivative you get by dividing the difference between CV averages by the distance between cell centers. That is definitely not the normal derivative that we need for this flux, and for highly skewed meshes will give ridiculously bad answers. Don't ever use this directional derivative for anything.

We can apply this interpretation to find the maximum time step for multi-dimensional unstructured meshes. For a given cell, we first compute the integral around the cell of the *incoming* flux for an assumed value of $a_0 = 1$. This means, in practice for one dimension,

$$\left(\sum\right)_i = \max\left(c_{i-\frac{1}{2}}, 0\right) - \min\left(c_{i+\frac{1}{2}}, 0\right)$$

Then we take this sum (which is the *incoming* flux with $a \equiv 1$) and divide it into the cell size to get the maximum time step, which reduces in the case of constant c to Equation 2.

The question of course is how do we extend this to two-dimensional unstructured meshes? Again, we use the concept of computing the integral of the *incoming* flux for the control volume:

$$\frac{1}{\Delta t_{\max;i}} = \frac{\oint_i \min\left(\vec{u}\cdot\hat{n}, 0\right)\, ds}{A_i}$$

where we evaluate the integral in exactly the same way we evaluate the flux integral.

This approach can be generalized in a straightforward way to systems, such as the Euler equations for compressible flow, by choosing the largest incoming velocity for each face in computing the maximum time step for the control volume.

Once the maximum stable time step is known for each control volume, the maximum stable time step for the entire mesh can be found be taking the minimum of these. That is,

$$\Delta t_{\max} = \min_i \Delta t_{\max,i}$$

7 Solution Reconstruction

This section will give a short overview of the reconstruction process for linear reconstruction.

Our goal can be stated relatively simply in words. We are given a set of piecewise constant data that we know approximates some smooth function. We want to find a polynomial approximation of degree k (in this case, a linear one, so $k = 1$) within each control volume so that our linear approximation matches that smooth function to order h^{k+1}, where h is the characteristic cell size. To enforce global conservation, we also require that the average of our polynomial approximation over the control volume must match the original control volume average.

Now let's state that mathematically.

Consider a smooth function $\phi(x, y)$. We don't know this function, but we do know its control volume averages $\overline{\phi}_i$ over the control volumes of an unstructured mesh.

We wish to compute a linear approximation to ϕ in control volume i, which we can write as

$$\phi_i^R(x,y) = \phi_i + \frac{\partial \phi}{\partial x}_i (x - x_i) + \frac{\partial \phi}{\partial y}_i (y - y_i)$$

ϕ_i^R is our linear approximation to the solution within control volume i. The right hand side is a truncated Taylor series expansion about the control volume reference location (x_i, y_i); in effect, we're setting up a coordinate system centered at that locationϕ_i, $\frac{\partial \phi}{\partial x}_i$, and $\frac{\partial \phi}{\partial y_i}$ are estimates (to be computed below) for the value and gradient of the function ϕ at the reference point of control volume i. The location of the reference point is, in principle, arbitrary. Typically, for vertex-centered control volumes, the easiest choice is to use the vertex location; for cell-centered control volumes, the centroid is the easiest choice.

The constraint that the average of our reconstructed function match the control volume average can be written as:

$$\begin{aligned}
\overline{\phi}_i &= \frac{1}{A_i} \int_i \phi_i^R(x,y)\, dx\, dy \\
&= \frac{1}{A_i} \int_i \phi_i + \frac{\partial \phi}{\partial x}_i (x - x_i) + \frac{\partial \phi}{\partial y}_i (y - y_i)\, dx\, dy \\
&= \phi_i + \frac{1}{A_i} \frac{\partial \phi}{\partial x}_i \int_i x - x_i\, dx\, dy + \frac{1}{A_i} \frac{\partial \phi}{\partial y}_i \int_i y - y_i\, dx\, dy \\
&= \phi_i + \frac{\partial \phi}{\partial x}_i \overline{x}_i + \frac{\partial \phi}{\partial y}_i \overline{y}_i
\end{aligned} \tag{3}$$

where in the last line, we take advantage of the definition of the centroid location $\overline{x}_i \equiv \frac{1}{A_i} \int_i x - x_i\, dx\, dy$ and likewise for $\overline{y}_i$.From Eqn. 3, it's immediately clear why the centroid is a particularly convenient choice of the reference location (x_i, y_i): this makes $\overline{x}_i$ and $\overline{y}_i$ both equal to zero.

If our reconstructed function is perfectly accurate, it will exactly predict the control volume averages in nearby control volumes j that are in the reconstruction stencil for i; more on choosing these control volumes later. Mathematically, we can express

this as:

$$\begin{aligned}
\overline{\phi}_i &= \frac{1}{A_j}\int_j \phi_i^R(x,y)\,dx\,dy \\
&= \frac{1}{A_j}\int_j \phi_i + \frac{\partial\phi}{\partial x}_i(x-x_i) + \frac{\partial\phi}{\partial y}_i(y-y_i)\,dx\,dy \\
&= \phi_i + \frac{1}{A_j}\frac{\partial\phi}{\partial x}_i\int_j (x-x_j+x_j-x_i)\,dx\,dy + \frac{1}{A_j}\frac{\partial\phi}{\partial y}_i\int_j (y-y_j+y_j-y_i)\,dx\,dy \\
&= \phi_i + \frac{1}{A_j}\frac{\partial\phi}{\partial x}_i\int_j (x-x_j)\,dx\,dy + \frac{1}{A_j}\frac{\partial\phi}{\partial y}_i\int_j (y-y_j)\,dx\,dy + \frac{\partial\phi}{\partial x_i}(x_j-x_i) + \frac{\partial\phi}{\partial y_i}(y_j-y_i) \qquad (4) \\
&= \phi_i + \frac{\partial\phi}{\partial x}_i(\overline{x}_j+x_j-x_i) + \frac{\partial\phi}{\partial y}_i(\overline{y}_j+y_j-y_i) \qquad (5)
\end{aligned}$$

In the third line, we added zero in a convenient form, so that the next line contains the moments of control volume j about its own reference point, plus the difference between reference locations. This makes it so that we only ever need to calculate moments for any control volume about its *own* reference point, making it just one integral per control volume, instead of one integral for every stencil the control volume appears in. We can abbreviate

$$\begin{aligned}
\Delta x_{ij} &= x_j - x_i \\
\Delta y_{ij} &= y_j - y_i
\end{aligned}$$

Now, if we happen to have exactly as many cells in the reconstruction stencil for control volume i as there are derivatives in our Taylor series expansion, we can write down a linear system that we can solve for ϕ_i, $\frac{\partial\phi}{\partial x}_i$, and $\frac{\partial\phi}{\partial y_i}$. In practice, though, we want more control volumes in the stencil than derivatives for stability reasons. Let's take the specific example of a 2D triangular mesh, using cell-centered control volumes. In this case, the most obvious stencil choice for linear reconstruction is to use the control volumes that share an edge with the one where we're doing reconstruction; away from boundaries, there will be three of these, which we'll label P, Q, and R. When we write down the mean constraint for control volume i (Eq. 3) and the predictions for the control volume averages in P, Q, and R (three copies of Eq. 5) as a single system of equations, we get:

$$\left[\begin{array}{c|cc} 1 & \overline{x}_i & \overline{y}_i \\ \hline 1 & \overline{x}_P + \Delta x_{iP} & \overline{y}_P + \Delta y_{iP} \\ 1 & \overline{x}_Q + \Delta x_{iQ} & \overline{y}_Q + \Delta y_{iQ} \\ 1 & \overline{x}_R + \Delta x_{iR} & \overline{y}_R + \Delta y_{iR} \end{array}\right] \begin{pmatrix} \phi_i \\ \frac{\partial\phi}{\partial x}_i \\ \frac{\partial\phi}{\partial y}_i \end{pmatrix} = \begin{pmatrix} \overline{\phi}_i \\ \hline \overline{\phi}_P \\ \overline{\phi}_Q \\ \overline{\phi}_R \end{pmatrix}$$

This system of equations can't be solved exactly, because there are more equations than unknowns. The mean constraint (above the horizontal line) must be satisfied exactly, while the others do not need to be exact. We can use Gauss elimination to zero out the first column (below the first row):

$$\left[\begin{array}{ccc} 1 & \overline{x}_i & \overline{y}_i \\ \hline 0 & \hat{x}_{iP} & \hat{y}_{iP} \\ 0 & \hat{x}_{iQ} & \hat{y}_{iQ} \\ 0 & \hat{x}_{iR} & \hat{y}_{iR} \end{array}\right] \begin{pmatrix} \phi_i \\ \\ \frac{\partial \phi}{\partial x}_i \\ \\ \frac{\partial \phi}{\partial y}_i \end{pmatrix} = \begin{pmatrix} \overline{\phi}_i \\ \hline \overline{\phi}_P - \overline{\phi}_i \\ \overline{\phi}_Q - \overline{\phi}_i \\ \overline{\phi}_R - \overline{\phi}_i \end{pmatrix} \tag{6}$$

where

$$\hat{x}_{ij} = \overline{x}_j + \Delta x_{ij} - \overline{x}_i$$
$$\hat{y}_{ij} = \overline{y}_j + \Delta y_{ij} - \overline{y}_i$$

We separate out the last three equations:

$$\begin{bmatrix} \hat{x}_{iP} & \hat{y}_{iP} \\ \hat{x}_{iQ} & \hat{y}_{iQ} \\ \hat{x}_{iR} & \hat{y}_{iR} \end{bmatrix} \begin{pmatrix} \frac{\partial \phi}{\partial x}_i \\ \\ \frac{\partial \phi}{\partial y}_i \end{pmatrix} = \begin{pmatrix} \overline{\phi}_P - \overline{\phi}_i \\ \overline{\phi}_Q - \overline{\phi}_i \\ \overline{\phi}_R - \overline{\phi}_i \end{pmatrix} \tag{7}$$

and solve these using any of several methods for least-squares problems: normal equations, QR decomposition, singular-value decomposition (SVD), etc. The normal equation approach is really easy, but has problems with numerical conditioning (small numerical errors have a large effect on the final result). My group uses SVD exclusively, because it's a robust method, it gives some indication how things have gone wrong when they do, and it's possible to pre-compute things so that at run time, we can compute

$$\begin{pmatrix} \frac{\partial \phi}{\partial x}_i \\ \\ \frac{\partial \phi}{\partial y}_i \end{pmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{pmatrix} \overline{\phi}_P - \overline{\phi}_i \\ \overline{\phi}_Q - \overline{\phi}_i \\ \overline{\phi}_R - \overline{\phi}_i \end{pmatrix}$$

where the matrix on the RHS comes from computing the SVD. Once we have the derivatives, it's easy to compute

$$\phi_i = \overline{\phi}_i - \frac{\partial \phi}{\partial x}_i \overline{x}_i - \frac{\partial \phi}{\partial y}_i \overline{y}_i$$

Now we have a polynomial representation of the solution that we can evaluate at flux integration points to get the data we need to compute fluxes. This reconstruction is second-order accurate (and will give an exact reconstruction for linear initial data),

and fluxes that we calculate based on this data will be second-order accurate. However, the reconstruction is not continuous at control volume boundaries. That is, if we compute $\phi_i(x_a, y_a)$ and $\phi_j(x_a, y_a)$, where i and j are adjacent control volumes, and (x_a, y_a) is some point on the boundary between them, we'll find that

$$\phi_i(x_a, y_a) - \phi_j(x_a, y_a) = \mathcal{O}\left(h^2\right)$$

That is, both reconstructions are within $\mathcal{O}\left(h^2\right)$ of the exact solution, but the constants on the error are different, and so the two reconstructions also differ from each other by $\mathcal{O}\left(h^2\right)$.

In the structured mesh context, we often combined reconstruction and evaluation, by computing

$$\begin{aligned} T_{i+\frac{1}{2}} &= \frac{3}{2}\overline{T}_i - \frac{1}{2}\overline{T}_{i-1} \\ &= \overline{T}_i + \frac{\Delta x}{2}\left(\frac{\overline{T}_i - \overline{T}_{i-1}}{\Delta x}\right) \end{aligned}$$

where the quantity in parentheses is the approximation to the derivative, and the whole RHS is an evaluation of that linear approximation.

8 Second-Order Accurate Flux Evaluation

> The compressible flow homework assignment will require you to compute a second-order flux integral for the advection terms in Eq. 1.

8.1 Note on Viscous and Viscous-like Fluxes

Fluxes with gradients, like those in the heat equation and the viscous terms of the Navier-Stokes equations, can be evaluated using the same reconstruction data used for the inviscid terms. That reconstruction results in a Taylor series expansion for the solution within each control volume. For a second-order accurate solution, the reconstruction can be written explicitly as

$$\begin{aligned} T(x, y) = T(x_{\text{cent}}, y_{\text{cent}}) + \frac{\partial T}{\partial x}\Delta x + \frac{\partial T}{\partial y}\Delta y \\ + O\left(\Delta x^2, \Delta x \Delta y, \Delta y^2\right) \end{aligned}$$

To get the gradients we need, we can simply extract the derivative terms from the reconstruction. These are constant across the control volume, but not necessarily between control volumes, so an average must be taken at the interface before computing the flux integral.

This isn't quite the whole story, though. Our reconstructions are imperfect, or we'd have the same reconstructed solution at points on the control volume boundaries when assessed on both sides. That small jump implies that we have our gradient estimate slightly wrong. To account for this, we add a *jump term* to diffusive fluxes to "gradient-ize" the jump. The overall diffusive flux at a point a on the boundary between control volumes i and j, for linear reconstruction, is

$$\frac{\partial \phi}{\partial n}_a \approx \frac{1}{2}\left(\nabla_i \phi + \nabla_j \phi\right) \cdot \hat{n}_a + \alpha \frac{\phi_R\left(x_a, y_a\right) - \phi_L\left(x_a, y_a\right)}{\left(\vec{x}_R - \vec{x}_L\right) \cdot \hat{n}_a}$$

where R denotes the right control volume (the one into which the normal is pointing) and L denotes the left control volume (the one out of which the normal is pointing). The jump term helps with stability and on especially regular meshes can even help with accuracy. The latter is easy to achieve for structured meshes (it's possible to increase the order of accuracy with such a term), but exceedingly difficult to achieve for unstructured meshes (you can make the constant somewhat better, but that's all, in practice).

8.2 Note on Higher-Order Accuracy

The reconstruction techniques described here and in the papers in the reference list can be applied to produce not just piecewise linear reconstructions, but also piecewise quadratic, cubic, etc. reconstructions. See [OGV02] for details about how reconstruction is done for high-order accuracy. This is neither the original nor even perhaps the definitive reference on this subject, but I think that it's more accessible than most of the alternative papers, and it uses the same notation as used here. For higher-order accuracy, care must be taken to ensure that the flux integral is accurate enough to exploit the accuracy of the flux calculation itself; see [OGNM09] for a catalog of pitfalls that is as complete as we could make it. We will return to this topic in Section 9.

For inviscid-type fluxes, the solution value at control volume boundaries is reconstructed as usual using this higher-degree polynomial. This approach is known to give correct high-order accurate fluxes.

For viscous-type fluxes, the gradient of the reconstruction is no longer constant, so we must differentiate the entire Taylor series expansion. The result is shown here for the cubic case.

$$\begin{pmatrix} \frac{\partial T}{\partial x} \\ \\ \frac{\partial T}{\partial y} \end{pmatrix} \approx \begin{pmatrix} T_x + \Delta x T_{xx} + \Delta y T_{xy} + \frac{\Delta x^2}{2} T_{xxx} \\ \quad + \Delta x \Delta y T_{xxy} + \frac{\Delta y^2}{2} T_{xyy} \\ \\ T_y + \Delta x T_{xy} + \Delta y T_{yy} + \frac{\Delta x^2}{2} T_{xxy} \\ \quad + \Delta x \Delta y T_{xyy} + \frac{\Delta y^2}{2} T_{yyy} \end{pmatrix}$$

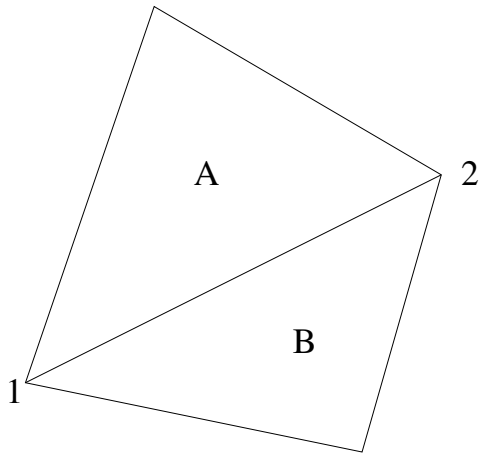


Figure 6: Face in an unstructured mesh, showing labeling of cells and vertices.

where all the derivatives on the right-hand side are evaluated at the centroid. In other words, these are the derivatives that the reconstruction automatically computes. Δx and Δy are distances from the control volume reference point to the flux evaluation point. The fluxes that we get from this are only third-order accurate, while the reconstruction and the inviscid fluxes are fourth-order accurate.

9 Detailed Anatomy of an Unstructured Mesh Flow Solver

9.1 Data structures

With the flexibility of unstructured meshes comes the responsibility of keeping track explicitly of the topology and geometry of the mesh. Needless to say, there is a fair bit of data that must be stored. The data that my cell-centered flow solver stores is:

- Mesh size data: the number of vertices, edges, cells, boundary edges, and control volumes.
- For each vertex, the coordinates of the vertex.
- For each face, the vertices at the end points of the face and the cells on each side. These are stored in a way that retains orientation information, as shown in Figure 6.
- For each cell, several things must be stored.
  - The location of the centroid.

  - A list of nearby cells to provide information for reconstruction. This list is created as a pre-processing step, in which successive layers of cells are added to provide enough data for reconstruction, as outlined in my reconstruction papers.
  - The moments of the cell are stored, because these are required by the reconstruction. These are computed in a one-time geometric pre-processing step. During this pre-processing, we need to calculate moments around the control volume centroid; each moment is of the form

$$M_{kl} = \frac{1}{A} \int_{CV} x^k y^l \, dx \, dy$$

    These can be evaluated either by direct integration over the control volume or by using Gauss's Theorem to transform them into integrals of the form

$$M_{kl} = \frac{1}{A} \oint_{\partial CV} \frac{x^{k+1} y^l}{k+1} \, ds$$

    The latter form is more convenient to use when curved boundaries are present, because the contour integration can be done nearly as easily along a curved cell boundary as a straight one.

- For each boundary face, information about the boundary condition and the physical mesh face is stored. Also, for high-order schemes, information about the location of and normals at flux integration points are given, because the boundary may be curved.

A C data structure that holds all of this information is shown in the following code fragment. `NDIM` is the number of space dimensions of the problem.

```
struct __mesh {
  /**** Mesh size data ****/
  int iNVerts, iNEdges, iNBdryEdges;
  int iNCells, iNCV;

  /**** Vertex data ****/

  /* Vertex coordinates */
  double (*a2dCoord)[NDIM];

  /**** Edge data ****/
```

```
  /* Edge- and cell-to-vert
     connectivity */
  int (*a2iEdgeVert)[2];
  /* Edge-to-cell connectivity */
  int (*a2iEdgeCell)[2];

  /**** Cell data ****/

  /* Cell centroid coordinates */
  double (*a2dCent) [NDIM];
  /* Neighbor data for
     reconstruction */
  int *aiNPts2, *aiNPts3, *aiNPts4;
  int *aiNeighList, *aiNeighStart;
  /* Control volume moments */
  double (*a2dMoments)[10];

  /**** Boundary edge data ****/

  /* Integration points for each
     boundary edge */
  int iNGaussPts, iNBdryGaussPts;
  /* Boundary edge integration info:
     Gauss point locations, normals
     and weights. */
  double
    (*a3dBdryGaussPts)[2][NDIM],
    (*a2dBdryGaussWts)[2],
    (*a3dBdryNormal)[2][NDIM];
  /* Mark which boundary edge goes
     with which control volume */
  int (*a2iBdryCV)[2];
  /* Mark a boundary condition for
     each edge */
  int (*aiBdryCond);
};

typedef struct __mesh Mesh;
```

9.2 Single pass over edges for flux integration

This subsection will describe the technique for computing the flux integral for each control volume in the mesh.

The obvious-but-not-so-efficient way of doing this would be to integrate around one cell, then move to the next, and so on. The reason that this isn't very efficient is that the flux from cell A to cell B in Figure 6 must be calculated twice, not just once. As we'll see for compressible flow, flux functions can get to be expensive, so this is a bad idea.

To get around this, we can instead evaluate the fluxes only once per face and then apply this flux to both neighboring flux integrals. The overall procedure goes like this for first- and second-order accurate schemes. If you're dying of curiosity about how to do this for higher-order schemes, come see me.

First, we perform reconstruction in every control volume. Then, for every interior face...

1. Find the location of the center of the face, the length of the face, and the normal to the face. This is done in the same way that we did for structured meshes. Referring to Figure 6, I choose the normal to point into cell B, so the normal is
$$\hat{n} = \frac{1}{l_{12}} \begin{pmatrix} y_2 - y_1 \\ \\ -(x_2 - x_1) \end{pmatrix}$$

2. Evaluate the flux at the center of the face. For fluxes that depend on the solution, this requires that I use the (previously calculated) reconstruction of the solution to find the approximate value of the solution on each side of the interface at the midside and apply a flux function to this data to find the numerical flux. For fluxes that depend on the gradient, the gradient is evaluated at the midside. In both cases, the flux will depend on the direction of the normal $\hat{n}$.

3. Multiply the flux by the edge length. Add the result to the flux integral in cell B and subtract it from the flux integral in cell A.

For boundary faces, the idea is the same, but the flux is computed differently depending on what the boundary condition is.

The concept really is that easy, believe it or not. As with any other CFD program, the details must nearly all be right before the program begins to look like it's working and *all* of the details must be right before the program really does work.

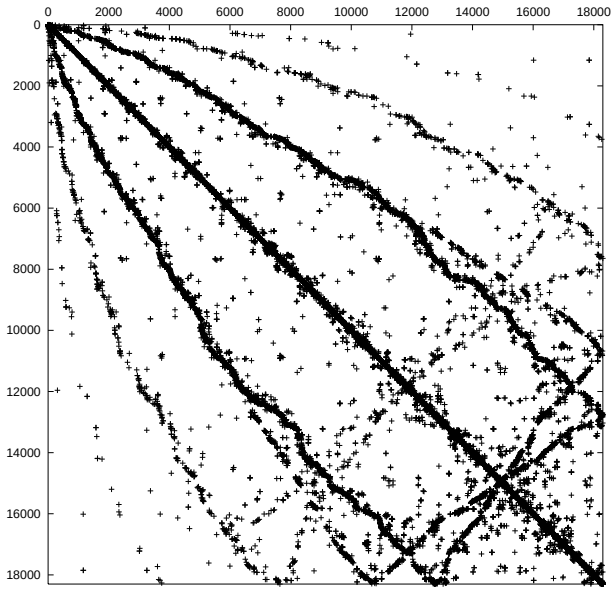


Figure 7: Matrix fill for unstructured mesh as-generated.

10 Intro to Unstructured Implicit Methods

In principle, one can form an implicit left-hand side matrix for an unstructured mesh computation in the same way as for a structured mesh computation: by taking the Jacobian of the flux integral for each control volume with respect to the solution in each control volume. There are two challenges with this in practice. First, the structure of the matrix makes solution of the system of equations challenging. Second, forming the matrix at all is a challenge.

10.1 Matrix structure

For a typical unstructured mesh, you might get a pattern of non-zeroes in the implicit LHS matrix that looks something like Figure 7. This matrix would, inevitably, be stored in some sparse matrix format that takes advantage of the huge fraction of zeroes to reduce memory requirements, so that isn't an issue. And Krylov methods can be applied to the matrix as well. The issue is that preconditioning requires an approximate inverse, typically by using ILU or some other approximate factorization technique, which is quite expensive computationally when the matrix bandwidth is high.

To improve on the bandwidth, a common choice is to use reverse Cuthill-McKee

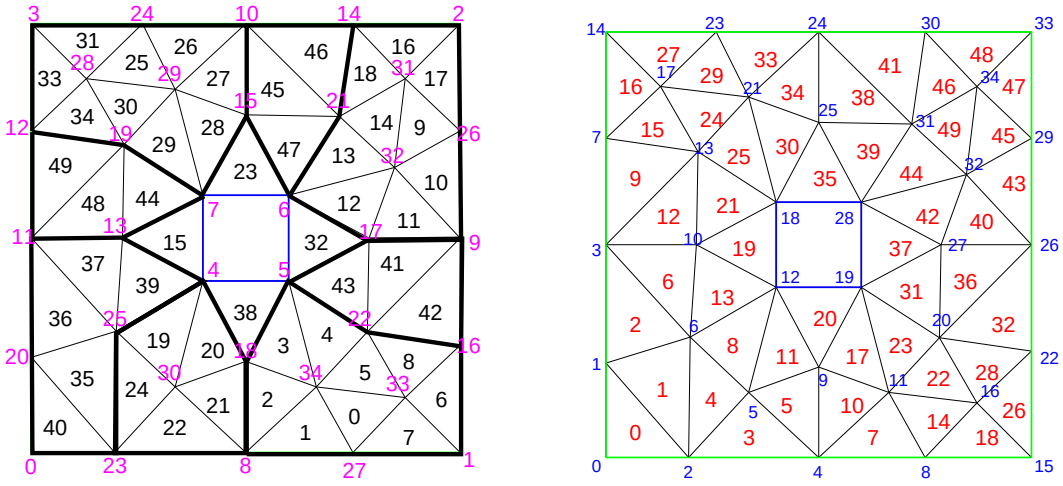


Figure 8: Before and after pictures of mesh re-ordering for a small unstructured mesh.

ordering for the vertices. The idea here is to begin with a vertex that has a small number of neighbors and label this as vertex 1. Then its neighbors are listed in order from lowest to highest degree (number of neighbors) and added to the list starting at 2. Then their neighbors are added, and so on, until all the vertices have been renumbered. At the end, the order of the list is reversed; this has been shown heuristically to be helpful for reducing fill in ILU decomposition.

To reorder cells (faces), one can apply RCM directly to the cells (faces) or renumber based on vertex ordering. In the latter case, cells (faces) are sorted on the basis of the sum of the indices of their vertices. This makes cell (face) ordering track vertex ordering reasonably well, which helps improve data locality and cache hit rate.

For a small, simple mesh, the effects of re-ordering are shown in Figure 8. In the "before" picture, the dark lines separate regions where cells are numbered more or less sequentially. Finally, Figure 9 shows the matrix fill for a large mesh after reordering. This result is typical of the bandwidth that's achievable in general; the bandwidth is proportional to $\sqrt{N}$, where N is the number of control volumes.

10.2 Computing the entries in the LHS

The real challenge for unstructured implicit methods, though, is computing the entries to the LHS matrix. The reason this is so hard is that, in principle, one must include the influence of all the control volumes in the reconstruction for a given control volume and all of its neighboring control volumes to get the correct entries

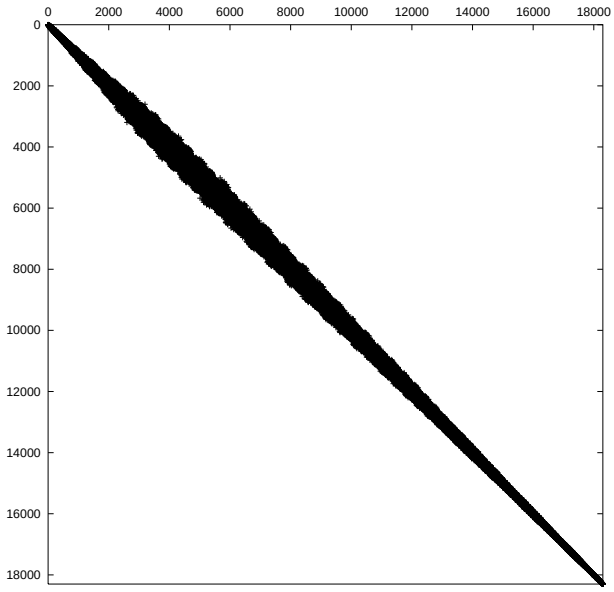


Figure 9: Matrix fill for unstructured mesh after re-ordering.

on a given row. This is pretty daunting, even before you think about limiters in the reconstruction and so on.

One way around much of this problem is to use a matrix-free Krylov method. The idea here is pretty simple: a Krylov method for a CFD problem can be written schematically as:

$$\left(\frac{I}{\Delta t} - \frac{\partial \mathrm{FI}_i}{\partial \overline{U}_j}\right) \delta U_j = \mathrm{FI}_i$$

where the entire mass of Jacobian matrices on the LHS has been compressed into a derivative of the flux integral. Now, remember that the LHS matrix is used in Krylov methods only to compute the matrix-vector product $\left(\frac{I}{\Delta t} - \frac{\partial \mathrm{FI}}{\partial \overline{U}}\right) v_k$. The hard part of this term to compute is simply the directional derivative of the flux integral in the direction of v_k. So we can compute this from the definition of a derivative:

$$\frac{\partial \mathrm{FI}}{\partial \overline{U}} v_k \approx \frac{\mathrm{FI}\left(\bar{U} + \varepsilon v_k\right) - \mathrm{FI}\left(\overline{U}\right)}{\varepsilon}$$

where a good choice for ε is the square root of the machine precision (so $\varepsilon \approx 10^{-8}$ is a good choice for double-precision calculations).

This technique makes it possible to compute the product $\frac{\partial \mathrm{FI}}{\partial \overline{U}} v_k$ for essentially the cost of a flux integral evaluation, and makes it relatively simple to create a series

of basis vectors v_k. The catch is that you still need to precondition this problem with some approximate inverse of $\frac{\partial \mathrm{FI}}{\partial \overline{U}}$. This is often done using the LHS for a simpler discretization. There also are some advanced approaches that apply the preconditioner without actually computing an approximation to the LHS at all.

References

[Bar92] Timothy J. Barth. Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations. In *Unstructured Grid Methods for Advection-Dominated Flows*, pages 6–1 – 6–61. AGARD, Neuilly sur Seine, France, 1992. AGARD-R-787.

[Bar93] Timothy J. Barth. Recent developments in high order k-exact reconstruction on unstructured meshes. AIAA paper 93-0668, January 1993.

[CM69] E. H. Cuthill and J. M. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 24th National Conference of the Association for Computing Machinery*, pages 157–172, 1969.

[MOG07] Christopher Michalak and Carl Ollivier-Gooch. Matrix-explicit GMRES for a higher-order accurate inviscid compressible flow solver. In *Proceedings of the Eighteenth AIAA Computational Fluid Dynamics Conference*, 2007.

[NOG08] Amir Nejat and Carl Ollivier-Gooch. A high-order accurate unstructured finite volume Newton-Krylov algorithm for inviscid compressible flows. *Journal of Computational Physics*, 227(4):2592–2609, 2008.

[OG97] Carl F. Ollivier-Gooch. Quasi-ENO schemes for unstructured meshes based on unlimited data-dependent least-squares reconstruction. *Journal of Computational Physics*, 133(1):6–17, 1997.

[OGNM09] Carl Ollivier-Gooch, Amir Nejat, and Christopher Michalak. On obtaining and verifying high-order finite-volume solutions to the Euler equations on unstructured meshes. *American Institute of Aeronautics and Astronautics Journal*, 47(9):2105–2120, 2009.

[OGV02] Carl F. Ollivier-Gooch and Michael Van Altena. A high-order accurate unstructured mesh finite-volume scheme for the advection-diffusion equation. *Journal of Computational Physics*, 181(2):729–752, sep 2002.