Finite Volume Methods on Unstructured Meshes

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.

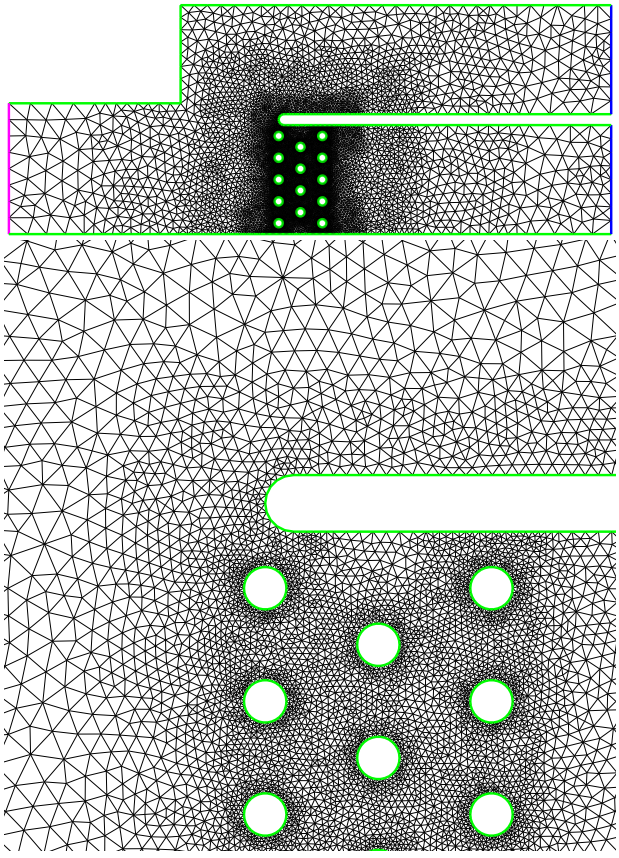

Figure 1: Branched-duct mesh, and close-up near splitter plate.

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 steadily growing in popularity, 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 2 is an unstructured mesh 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 3 is structured. So what is the distinction? I prefer the following definition.

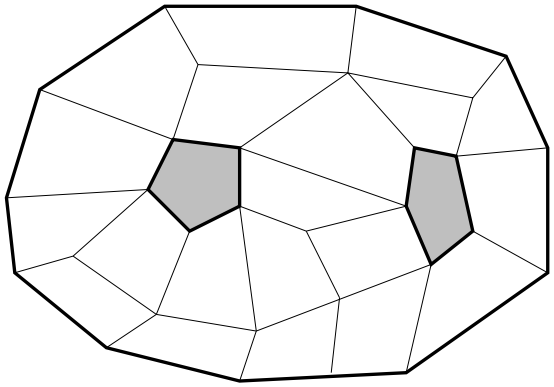

Figure 2: Unstructured quadrilateral mesh.

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 3, 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.

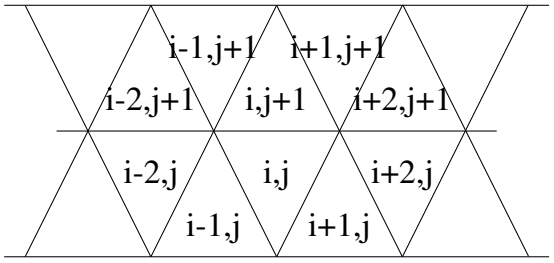

Figure 3: Structured triangular mesh.

1.1 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. Personally, I choose to store the following information when doing mesh generation:

- For each face, the identity of its vertices and the cells on either side of it.
- For each cell, the identity of its bounding faces.
- For each vertex, one face which contains it.
- For boundary faces, some additional information regarding boundary conditions and so forth.

This is actually 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.

1.2 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 4. We'll focus on cell-centered control volumes for geometric simplicity. Also, we will do

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

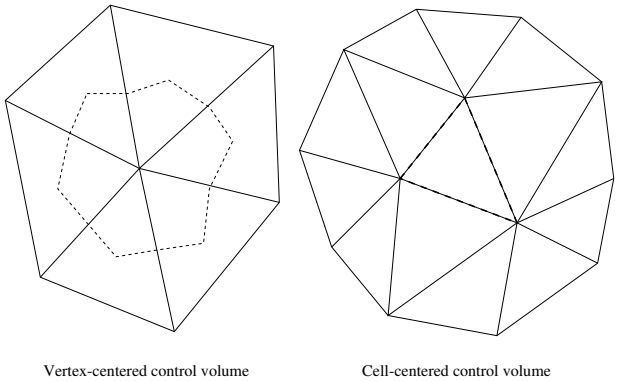

Figure 4: Control volumes for unstructured triangular meshes.

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.

2 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 equation on a triangular mesh. That is, we will solve the following problem:

$$\frac{\partial a}{\partial t} + \frac{\partial au(x,y)}{\partial x} + \frac{\partial av(x,y)}{\partial y} = 0 \qquad (1)$$

on the domain shown in Figure 5. The boundary conditions are as follows:

- D is an inflow boundary, where a is specified.
- Along segments A and C, the velocity perpendicular to the wall is zero.

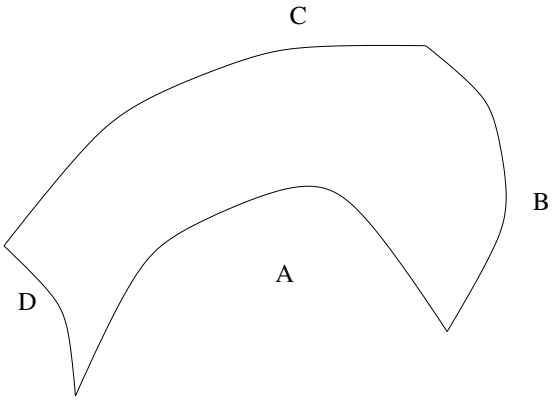

Figure 5: Geometry for two-dimensional advection problem.

- Along segment B, the outflow, we will not need a boundary condition, as we will be computing fluxes using an upwind scheme.

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 the basic concepts that are needed to solve this problem, although there are some details that need consideration.

3 Outline of a Generic Finite-Volume Unstructured Mesh Scheme

For problems with no viscous-like fluxes, 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 the scheme will be:

1. Reconstruct the solution to the desired order of accuracy and use this information to calculate the fluxes.

2. For each face in the mesh, compute the integral of the flux across that face. This flux should be subtracted from the "donor" cell and added to the "receptor" cell. The result of this integration and summation will be the computation of the residual for each control volume, provided that boundary faces also have their flux integrals calculated (perhaps with a special formula). This residual can be written in the familiar mathematical form
$$R_i = \oint_{\partial CV_i} \vec{F} \cdot \vec{n}\, ds$$

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

4. Advance the solution in time using your favorite explicit scheme. A first-order explicit time advance scheme would be
$$U_i^{n+1} = U_i^n + \Delta t\, R_i^n$$
where
$$\Delta t = \mathrm{CFL} \cdot \Delta t_{\max}$$

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

Section 4. First-order accurate flux evaluation and integration, both in the interior and on the boundary.

Section 5. Determination of the largest stable explicit time step.

[2] Again, 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 harder to evaluate.

Section 6. Solution reconstruction (in outline only, although a separate handout will give you more details).

Section 7. Second-order accurate flux evaluation and integration, both interior and boundary.

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 8 describes the approach used in my group's research code.

4 Flux Evaluation and Integration (First Order)

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 6. 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

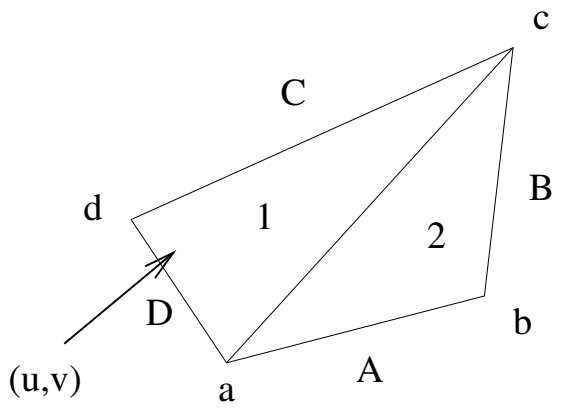

Figure 6: Control volumes for flux calculation

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 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 is positive, then flow is from 1 into 2, and vice versa, so we compute the flux as:

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

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).

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 6.[3] We know what the boundary fluxes

[3] Okay, so the mesh is a little too coarse. So what?

should be from the mathematical boundary conditions. The question is, how do we evaluate them in practice?

First, the two sides where there is no physical flux (A and C): here, we simply set the flux to zero. At the inflow and outflow, we use the upwind scheme much as we did in the interior. For face 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 face D, flow is into the domain. We 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 6 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) 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.

5 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 of Equation 2 the value in the cell at the next time step will not exceed a_0.

We can apply this interpretation to find the maximum time step. 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.

> **Exercise — Maximum Time Step**
>
> Given a velocity vector field $(u(x,y), v(x,y))$ and an arbitrary triangular control volume $\triangle abc$, find a formula for the maximum stable time step for the control volume.

This approach can be generalized in a straightforward way to systems, such as the Euler equations, 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,

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

6 Solution Reconstruction

The accompanying handout is a paper that describes how reconstruction is done. This this is neither the original nor even perhaps the definitive references on this subject, I think that it's more accessible than most of the alternative papers, and there are no copyright issues to worry about.

7 Second-Order Accurate Flux Evaluation

See homework assignment 1.

7.1 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

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

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.

7.2 Note on Higher-Order Accuracy

The reconstruction techniques described in the attached papers can be applied to produce not just piecewise linear reconstructions, but also piecewise quadratic, cubic, etc. reconstructions. 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. We will return to this topic in Section 8.

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.

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

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. The fluxes that we get from this are, in principle at least, only third-order accurate, while the reconstruction and the inviscid fluxes are fourth-order accurate. There is, however, some experimental evidence that suggests that the flux integration process cancels the leading error term, making the viscous fluxes fourth-order accurate in practice.

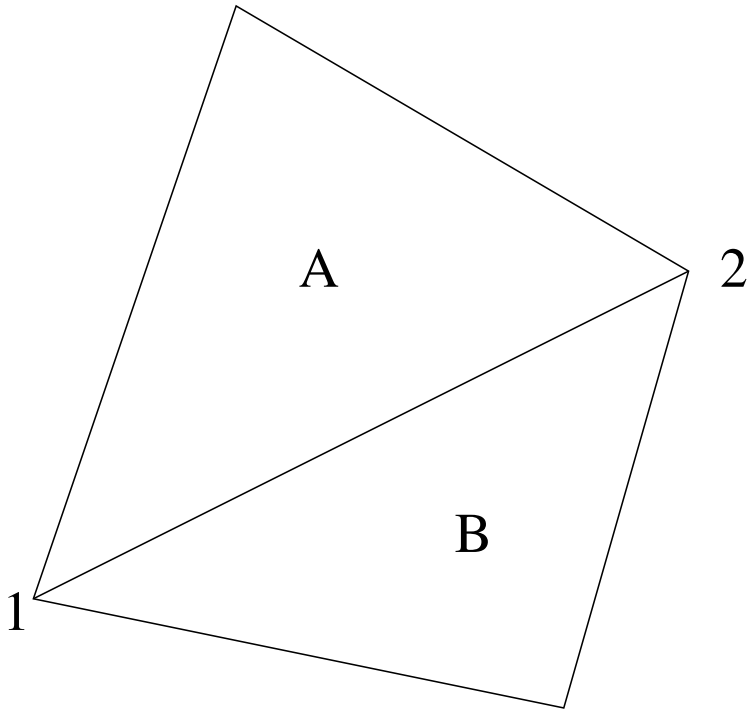

Figure 7: Face in an unstructured mesh, showing labeling of cells and vertices.

8 Detailed Anatomy of an Unstructured Mesh Flow Solver

8.1 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 7.
- 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 re-

construction. 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;
```

8.2 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 7 must be calculated twice, not just once.

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.

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 7, 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.

9 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.

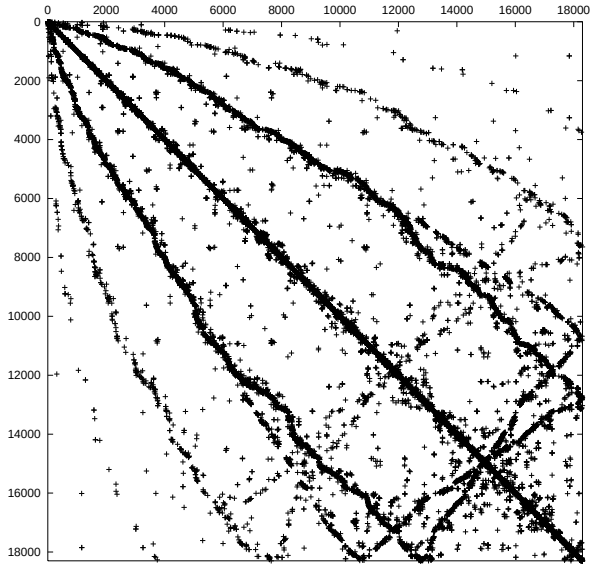

Figure 8: Matrix fill for unstructured mesh as-generated.

9.1 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 8. 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 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 reordering are shown in Figure 9. In the "before" picture, the dark lines separate regions where cells are numbered more or less sequentially. Finally, Figure 10 shows the matrix fill for a large mesh after reordering. This result is typical of the bandwidth that's achievable in general. My recollection (which may be incorrect) is that the bandwidth is proportional to $\sqrt{N}$, where N is the number of control volumes.

9.2 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 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

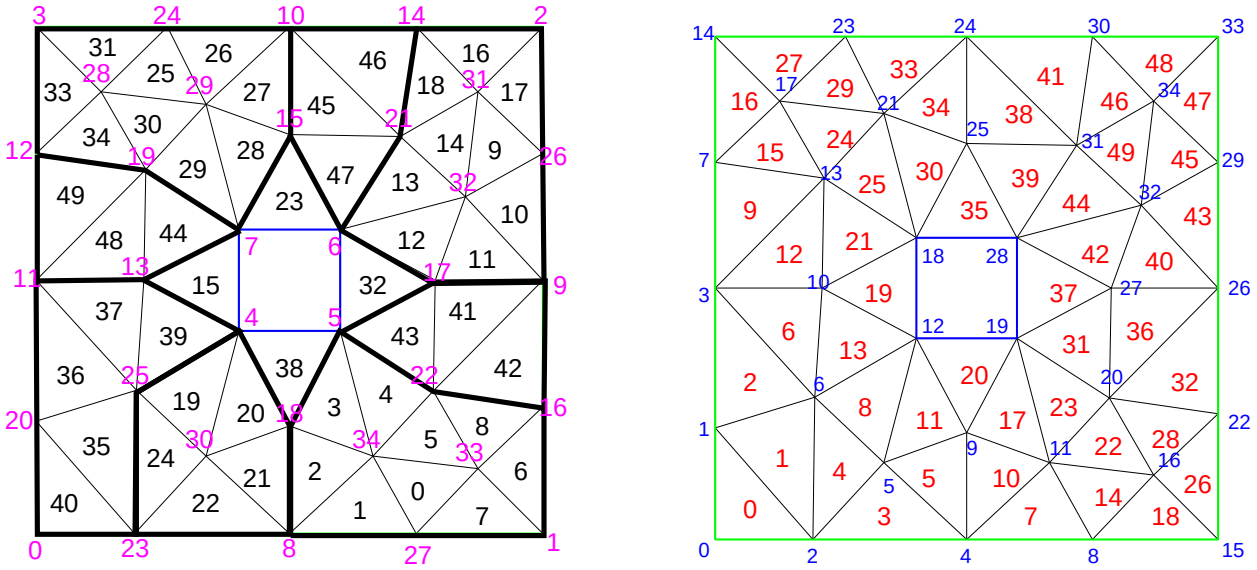

Figure 9: Before and after pictures of mesh re-ordering for a small unstructured mesh.

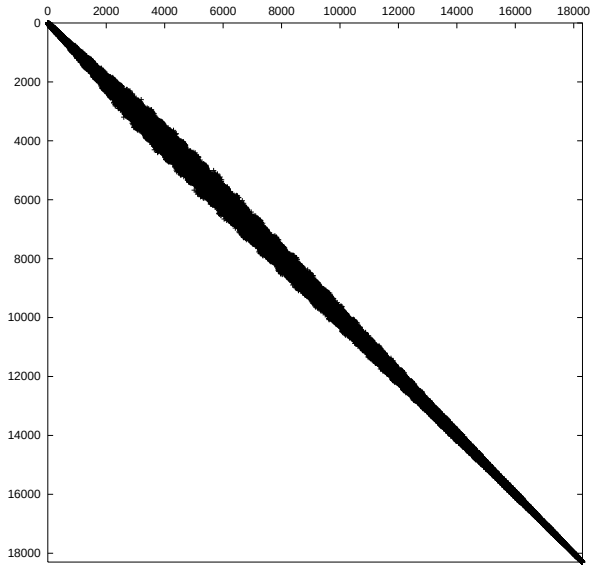

Figure 10: Matrix fill for unstructured mesh after re-ordering.

pretty simple: a Krylov method for a CFD problem can be written schematically as:

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

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 $\left(I - \frac{\partial \mathrm{FI}}{\partial \overline{U}}\right) v_k$. This term 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.

[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.