Lecture 41

Finite Elements

Triangulating a Region

A disadvantage of finite difference methods is that they require a very regular grid, and thus a very regular region, either rectangular or a regular part of a rectangle. Finite elements is a method that works for any shape region because it is not built on a grid, but on triangulation of the region, i.e. cutting the region up into triangles as we did in a previous lecture. The following figure shows a triangularization of a region.

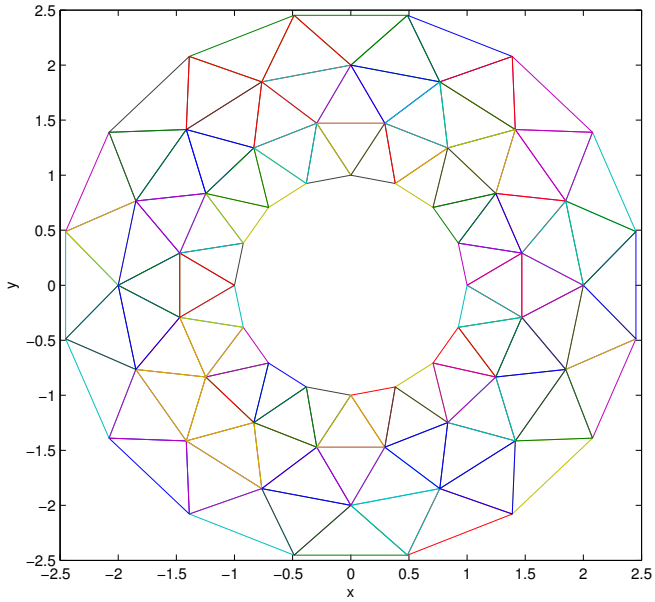

Figure 41.1: An annular region with a triangulation. Notice that the nodes and triangles are very evenly spaced.

This figure was produced by the script program `mywasher.m`. Notice that the nodes are evenly distributed. This is good for the finite element process where we will use it.

Open the program `mywasher.m`. This program defines a triangulation by defining the vertices in a matrix `V` in which each row contains the x and y coordinates of a vertex. Notice that we list the interior nodes first, then the boundary nodes.

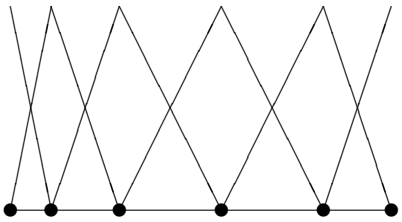

Figure 41.2: The finite elements for the "triangulation" of a one dimensional object.

Triangles are defined in the matrix `T`. Each row of `T` has three integer numbers indicating the indices of the nodes that form a triangle. For instance the first row is `43 42 25`, so T_1 is the triangle with vertices $\mathbf{v}_{43}$, $\mathbf{v}_{42}$ and $\mathbf{v}_{25}$. The matrix T in this case was produced by the MATLAB command `delaunay`. The command produced more triangles than desired and the unwanted ones were deleted.

A three dimensional plot of the region and triangles is produced by having the last line be `trimesh(T,x,y,z)`.

What is a finite element?

The finite element method is a mathematically complicated process. However, a finite element is actually a very simple object. To each node $\mathbf{v}_j$ we associate a function Φ_j that has the properties $\Phi_j(\mathbf{v}_j) = 1$ and $\Phi_j(\mathbf{v}_i) = 0$ for $i \neq j$. Between nodes, Φ_j is a linear function. This function is the finite element. (There are fancier types of finite elements that we will not discuss.)

If we consider one dimension, then a triangulation is just a subdivision into subintervals. In Figure 41.2 we show an uneven subdivision of an interval and the finite elements corresponding to each node.

In two dimensions, Φ_j is composed of triangular piece of planes. Thus Φ_j is a function whose graph is a pyramid with its peak over node $\mathbf{v}_j$.

What is a finite element solution?

A finite element (approximate) solution is a linear combination of the elements:

$$U(\mathbf{x}) = \sum_{j=1}^{n} C_j \Phi_j(\mathbf{x}). \tag{41.1}$$

Thus finding a finite element solution amounts to finding the best values for the constants $\{C_j\}_{j=1}^n$.

In the program `mywasher.m`, the vector `z` contains the node values C_j. These values give the height at each node in the graph. For instance if we set all equal to 0 except one equal to 1, then the function is a finite element. Do this for one boundary node, then for one interior node.

Notice that a sum of linear functions is a linear function. Thus the solution using linear elements is a piecewise linear function. Also notice that if we denote the j-th vertex by $\mathbf{v}_j$, then

$$U(\mathbf{v}_j) = C_j. \tag{41.2}$$

Thus we see that **the constants C_j are just the values at the nodes.**

Take the one-dimensional case. Since we know that the solution is linear on each subinterval, knowing the values at the endpoints of the subintervals, i.e. the nodes, gives us complete knowledge of the solution. Figure 41.3 could be a finite element solution since it is piecewise linear.

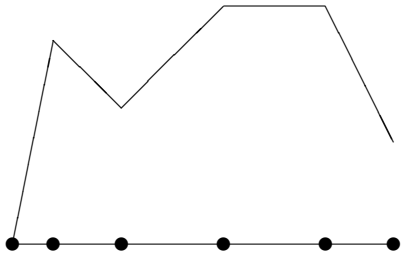

Figure 41.3: A possible finite element solution for a one dimensional object. Values are assigned at each node and a linear interpolant is used in between.

In the two-dimensional case, the solution is linear on each triangle, and so again, if we know the values $\{C_j\}_{j=1}^n$ at the nodes then we know everything.

Experiment with finite elements

By changing the values in `z` in the program we can produce different three dimensional shapes based on the triangles. The point then of a finite element solution is to find the values at the nodes that best approximate the true solution. This task can be subdivided into two parts: (1) assigning the values at the boundary nodes and (2) assigning the values at the interior nodes.

Values at boundary nodes

Once a triangulation and a set of finite elements is set up, the next step is to incorporate the boundary conditions of the problem. Suppose that we have fixed boundary conditions, i.e. of the form

$$u(\mathbf{x}) = g(\mathbf{x}) \quad \text{for} \quad \mathbf{x} \in \partial D,$$

where D is the object (domain) and ∂D is its boundary. Then the boundary condition directly determines the values on the boundary nodes.

In particular, suppose that $\mathbf{v}_\ell$ is a boundary node. Since $\Phi_\ell(\mathbf{v}_\ell) = 1$ and all the other elements are zero at node $\mathbf{v}_\ell$, then to make

$$U(\mathbf{v}_\ell) = \sum_{j=1}^{n} C_j \Phi_j(\mathbf{v}_\ell) = g(\mathbf{v}_\ell),$$

we must choose

$$C_\ell = g(\mathbf{v}_\ell).$$

Thus **the constants C_j for the boundary nodes are set at exactly the value of the boundary condition at the nodes.**

Thus, if there are m interior nodes, then

$$C_j = g(\mathbf{v}_j), \qquad \text{for all} \quad m+1 \le j \le n.$$

In the program `mywasher.m` the first 32 vertices correspond to interior nodes and the last 32 correspond to boundary nodes. By setting the last 32 values of `z`, we achieve the boundary conditions. We could do this by adding the following commands to the program:

```
z(33:64) = .5;
```

or more elaborately we might use functions:

```
z(33:48) = x(33:48).^2 - .5*y(33:48).^2;
z(49:64) = .2*cos(y(49:64));
```

Exercises

41.1 Generate an interesting or useful 2-d object and a well-distributed triangulation of it.

- Plot the region.
- Plot one interior finite element.
- Plot one boundary finite element.
- Assign values to the boundary using a function (or functions) and plot the region with the boundary values.

Turn in your code and the four plots.