Elliptic Grid Generation

Introduction

In this project you will use elliptic grid generation techniques to develop an "O" type grid for a two-dimensional airfoil. This exercise serves as an introduction to grid generation for odd geometries. The term "boundary-fitted coordinates" (BFC) is commonly used to describe such grids. These grids are used when the computational domain does not line up readily with a simple, e.g., Cartesian or polar, grid. In addition the project is also a somewhat advanced application of iterative methods used for solving elliptic equations. The equations to be solved are a coupled system of non-linear partial differential equations. Also one boundary condition will be taken to be periodic, rather than the fixed or derivative boundary conditions more frequently encountered with elliptic equations. The procedures discussed here are readily extended to three dimensions and to the use of Poisson rather than the Laplace equations as used here. A system of Poisson equations provides better control over interior spacing and orientation than what can be done with only Laplace equations.

Background

Elliptic grid generation is one of several methods used to generate structured grids for odd geometries. Algebraic methods are one commonly used alternative, and hyperbolic systems of equations are sometimes used, particularly for external flows. See, e.g., Thompson et al. (1985), Anderson et al. (1984), or Hoffman and Chiang (1993) for much more detailed information. In this project we will solve a pair of Laplace equations to generate the mesh. Rather than work in the physical domain where the geometry is complicated (which is why we are having to generate the grid) and the equations are simple, we prefer to work in the computational domain where just the opposite is true. Once we have done our work in the computational domain, we bring the results back to the physical domain for viewing.

When transformed from the physical (x, y) space to the computational (h,x) space, the Laplace equations $\nabla^2 h = 0$ and $\nabla^2 x = 0$ become:

$$a\, x_{xx} - 2b\, x_{xh} + g\, x_{hh} = 0 \tag{1}$$

$$a\, y_{xx} - 2b\, y_{xh} + g\, y_{hh} = 0 \tag{2}$$

where:

$$a = x_h^2 + y_h^2 \tag{3}$$

$$b = x_x x_h + y_x y_h \tag{4}$$

and

$$g = x_x^2 + y_x^2 \tag{5}$$

The boundaries in the physical region will be mapped to the computational plane to become boundary values for the two elliptic equations. The above equations will then be finite-differenced in the computational space and solved iteratively to fill in the interior values. The results will then be plotted up for inspection in the physical domain.

Implementation

The accompanying sketches of the physical and computational regions should help you with the mapping of the boundaries and, in particular, the handling of the periodic boundary at the trailing edge.

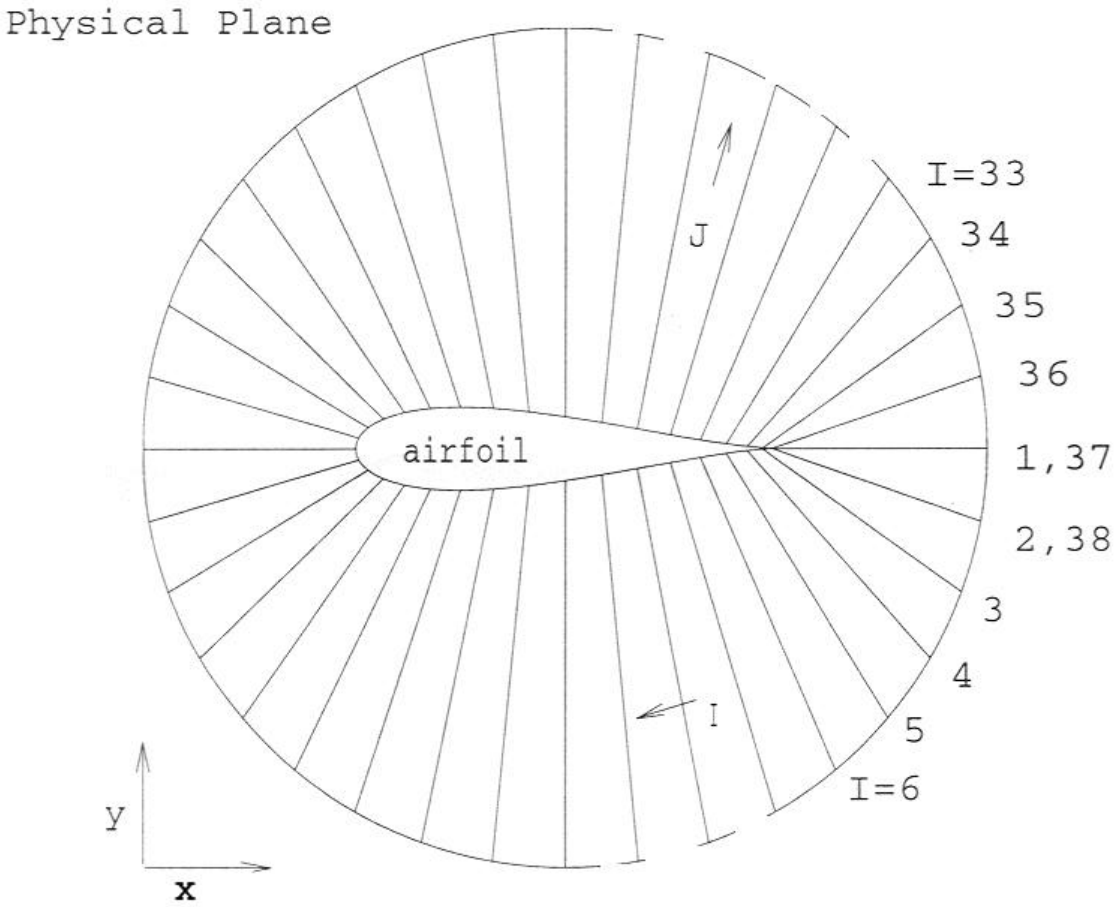

Figure 1 Physical Plane

The program FOILDR.FOR draws one particular airfoil in the physical plane and can be used to nodalize the airfoil boundary. Another option is to use tabulated x,y coordinate data for an airfoil of your choice. (See e.g., Abbott and von Doenhoff, 1959.) Whichever approach you choose, the x and y coordinate values then serve as boundary data along the lower boundary of the computational plane. The outer extreme in the physical plane is a circle of diameter perhaps two or three times the chord. (That is adequate here; in real life you would want a much bigger region.) These values need to be assigned along the upper boundary in the computational plane. At the "cut" emanating from the trailing edge, you can specify y = 0.0 for all points and linearly

distribute the x values between the trailing edge and the outer boundary. Once you have mapped the boundaries to the computational plane, use the TRANSF.FOR subroutine to fill in interior values using transfinite interpolation, a two-dimensional form of linear interpolation. Transfinite interpolation, gives a very good initial grid which then may be improved using the elliptic system. Besides the major time saving expected from using a good first "guess" with the iterative solution for the elliptic system, using transfinite interpolation first has another, very important side effect. The transformed equations are coupled and non-linear, and an iterative solution from just arbitrary initial conditions may converge to another, non-useable solution. (After you have your program working, you ought to try just that and see what you get.) You need to call the transfinite routine once for x and once for y (and should note in its internal documentation the requirement that you provide workspace for it in your main program). At this point you should plot this preliminary grid to make sure you have what you think you should have.

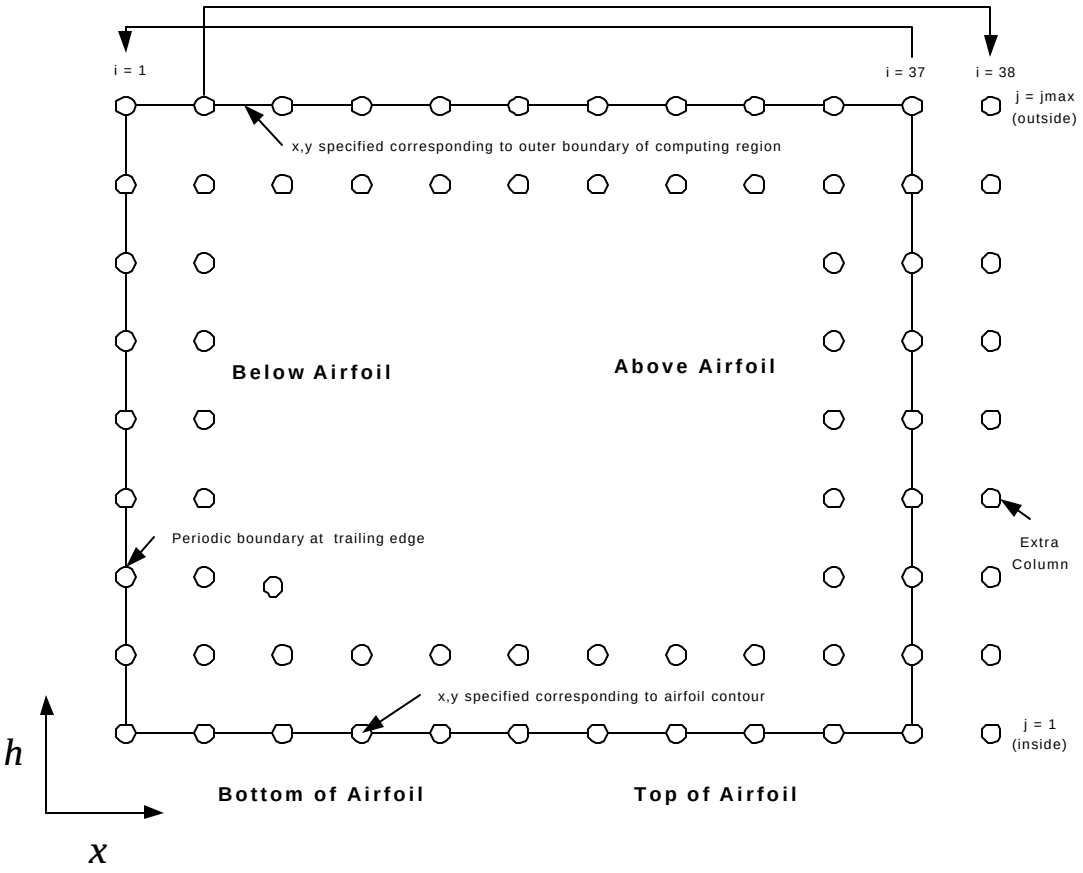

Figure 2 Computational Plane

As the third step the equations introduced earlier are to be finite-differenced using standard centered differences, and the resulting equations for x and y are to be solved using an iterative techniques. Gauss-Seidel iteration is easy to implement. The difference form of Equation 1 is solved easily for $x_{i,j}$ and that of Equation 2 is solved for $y_{i,j}$. Obviously the a, b, and g used here all have to be updated constantly as your iteration proceeds. The coding will consist of an outer loop controlling the number of overall sweeps and nested inner loops sweeping over the i and j indices. Within the innermost loop you will compute a, b and g as well as x

and y for the point being visited. Assuming that you use centered differences for all derivatives including the mixed partial, you will wind up with a nine-point stencil, rather than the more typical five-point.

For this assignment the values of x and y are considered to be fixed at j = 1, the airfoil surface and j = jmax, the outer boundary of the computing region. (There may actually be reasons to allow these points to be part of the solution.) A periodic boundary is to be implemented at the trailing edge, so that it acts essentially as part of the interior. A simple, frequently used method of implementing a periodic boundary is to add a fictitious extra column (corresponding in the figure to i = 38). The iteration runs from i = 2 to i = 37. At the end of a sweep the values of x and y at i = 37 are transferred to i = 1 (and used in the calculation at i = 2) and those at i = 2 are transferred the other way to i = 38 and used in the calculation at i = 37. This simple swap obviates the necessity of setting up separate forms of the difference equations at the ends.

Fortran 90 users will want to look into the use of the "Cshift" and "Eoshift" array functions as a means of implementing the periodic boundary conditions, as well as of addressing the neighbors making up the finite-difference "stencil".

After a sufficient number of sweeps (which does not need to be very large, because of the excellent first guess from the transfinite interpolation you are using first), plot your improved grid.

Probably your best development strategy is to forget the periodic boundary to start out with and implement it only after you have your program working reliably. That is, dimension only 37 x whatever corresponding to the schematic shown and assume that values of x and y are fixed for all values of j at i=1 and i=37. With these boundary conditions your grid should show a slope discontinuity at the cut and you will see why a periodic boundary is preferred.

An Alternative

As an alternative follow the same steps, but instead of an "O' grid, develop a "C" grid. See Thompson, et al. (1985) for details.

References

Abbott, I.H, and von Doenhoff, A.E., *Theory of Wing Sections*, Dover, New York, 1959.

Anderson, D.A., Tannehill, J.C., and Pletcher, R.H., *Computational Fluid Dynamics and Heat Transfer*, McGraw-Hill, New York (1984).

Hoffman, K.A. and Chiang, S.T., *Computational Fluid Dynamics for Engineers - Vols. I & II*, Engineering Education System, Wichita (1993).

Thompson, J.F., Warsi, Z.U.A. and Mastin, C.W., *Numerical Grid Generation - Foundations and Applications*, North-Holland, New York (1985).

GridGeneration 2/26/01