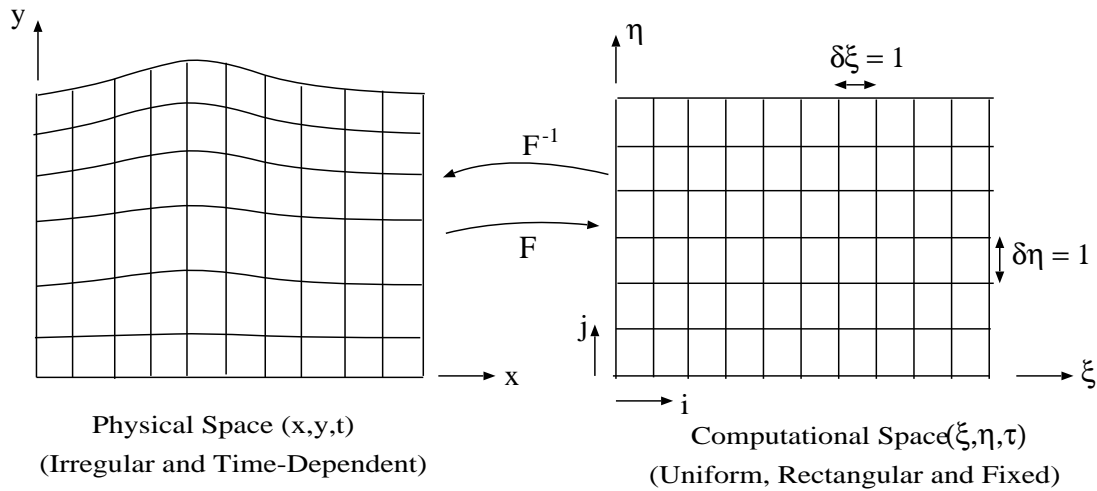


Generation of Boundary-Fitted Coordinates



So many methods have been developed for the generation of boundary-fitted coordinations (for the mapping F^{-1} illustrated in the figure) that that the topic has become a singular subject for even text books. As the present course has to address diverse topics on computational marine hydrodynamics in one semester, we shall discuss only one method here. Students interested to learn more of this topic may refer to literature on the subject or talk to the instructor.

One of the earliest methods developed for the generation of boundary-fitted coordinates is based on the Laplace equation:

$$\nabla_{xy}^2 \xi = 0; \quad \nabla_{xy}^2 \eta = 0$$

If the boundary-values are specified satisfying the Cauchy-Riemann conditions:

$$\xi_x = -\eta_y; \quad \xi_y = \eta_x,$$

which is not a trivial task or may not be possible at all for a given arbitrary boundary, one can expect the grids to be orthogonal akin to equi-potential and stream-function contours of a two-dimensional potential flow. If specification of boundary values can not satisfy the Cauchy-Riemann relation, one can nevertheless expect to generate reasonable (ie. nearly orthogonal) grids using the equations

$$\nabla_{xy}^2 \xi = 0; \quad \nabla_{xy}^2 \eta = 0$$

Applying the transformation relation for the Laplacian operator, obtained in the previous lecture, and noting the above equations to govern ξ and η variables on the (x, y, t) space, we get

$$\nabla_{xy}^2 x \equiv x_{xx} + x_{yy} = 0 = \nabla_{xy}^2 \xi \cdot x_\xi + \nabla_{xy}^2 \eta \cdot x_\eta + \frac{x_\eta^2 + y_\eta^2}{J^2} x_{\xi\xi} + \frac{x_\xi^2 + y_\xi^2}{J^2} x_{\eta\eta} - 2 \frac{x_\xi x_\eta + y_\xi y_\eta}{J^2} x_{\xi\eta}$$

Or, in view of $\nabla_{xy}^2 \xi = 0$ and $\nabla_{xy}^2 \eta = 0$,

$$\frac{x_\eta^2 + y_\eta^2}{J^2} x_{\xi\xi} + \frac{x_\xi^2 + y_\xi^2}{J^2} x_{\eta\eta} - 2 \frac{x_\xi x_\eta + y_\xi y_\eta}{J^2} x_{\xi\eta} = 0$$

Similar for y , one can obtain:

$$\nabla_{xy}^2 y \equiv y_{xx} + y_{yy} = 0 = \nabla_{xy}^2 \xi \cdot y_\xi + \nabla_{xy}^2 \eta \cdot y_\eta + \frac{x_\eta^2 + y_\eta^2}{J^2} y_{\xi\xi} + \frac{x_\xi^2 + y_\xi^2}{J^2} y_{\eta\eta} - 2 \frac{x_\xi x_\eta + y_\xi y_\eta}{J^2} y_{\xi\eta}$$

and since $\nabla_{xy}^2 \xi = 0$ and $\nabla_{xy}^2 \eta = 0$,

$$\frac{x_\eta^2 + y_\eta^2}{J^2} y_{\xi\xi} + \frac{x_\xi^2 + y_\xi^2}{J^2} y_{\eta\eta} - 2 \frac{x_\xi x_\eta + y_\xi y_\eta}{J^2} y_{\xi\eta} = 0$$

Denoting the coefficients as

$$A \equiv \frac{x_\eta^2 + y_\eta^2}{J^2}; \quad B \equiv \frac{x_\xi x_\eta + y_\xi y_\eta}{J^2}; \quad C \equiv \frac{x_\xi^2 + y_\xi^2}{J^2}$$

the equations to be used for obtaining (x, y) values on the nodes of the computational space can be written as

$$Ax_{\xi\xi} - 2Bx_{\xi\eta} + Cx_{\eta\eta} = 0$$

$$Ay_{\xi\xi} - 2By_{\xi\eta} + Cy_{\eta\eta} = 0$$

Note that the equations are nonlinear, elliptic partial-differential equations, but are tackled as quasi-linear equations by considering the equations as linear equations with variable coefficients. Having specified (x, y) values on the boundary nodes of the computational space, one has to then solve the above equations to determine (x, y) values on the nodes of the computational space.

The equations are solved numerically as follows. Let us consider only one of the two equations, as the steps are identical for both the equations. Upon discretizing the above governing equations, say by using central difference schemes, one obtains:

$$A(x_{i+1,j} + x_{i-1,j} - 2x_{i,j}) - \frac{B}{2}(x_{i+1,j+1} + x_{i-1,j-1} - x_{i+1,j-1} - x_{i-1,j+1}) + C(x_{i,j+1} + x_{i,j-1} - 2x_{i,j}) = 0$$

Note that $\delta\xi = \delta\eta = 1$. Keeping the $x_{i,j}$ term only on the left and moving the rest to the right, one can rewrite the above as

$$x_{i,j}(2A + 2C) = A(x_{i+1,j} + x_{i-1,j}) - \frac{B}{2}(x_{i+1,j+1} + x_{i-1,j-1} - x_{i+1,j-1} - x_{i-1,j+1}) + C(x_{i,j+1} + x_{i,j-1})$$

Or,

$$x_{i,j} = \frac{1}{2(A+C)} \left[A(x_{i+1,j} + x_{i-1,j}) - \frac{B}{2}(x_{i+1,j+1} + x_{i-1,j-1} - x_{i+1,j-1} - x_{i-1,j+1}) + C(x_{i,j+1} + x_{i,j-1}) \right]$$

Note that the subscripts i and j denote the nodes along ξ and η directions, respectively. The equation is then solved iteratively. The initial grid can be generated using simple algebraic interpolations.

Point-Jacobi Iteration Method. Denoting iteration counter with letter m in the superscript, the point Jacobi iterative scheme for the above equation can be written as

$$x_{i,j}^{m+1} = \frac{1}{2(A^m + C^m)} \left[A^m (x_{i+1,j}^m + x_{i-1,j}^m) - \frac{B^m}{2} (x_{i+1,j+1}^m + x_{i-1,j-1}^m - x_{i+1,j-1}^m - x_{i-1,j+1}^m) + C^m (x_{i,j+1}^m + x_{i,j-1}^m) \right]$$

The iteration is carried out until convergence. One may seek point-wise convergence as

$$|x_{i,j}^{m+1} - x_{i,j}^m| \leq \epsilon, \forall i, j$$

where ϵ is a user-specified small number. Point-wise convergence is not needed and one can instead seek l_2 type convergence as

$$\sum_{i=1}^{i=imax} \sum_{j=1}^{j=jmax} \frac{(x_{i,j}^{m+1} - x_{i,j}^m)^2}{(x_{i,j}^m)^2} \leq \epsilon$$

Gauss-Seidel Iteration Method. Gauss-Seidel method is an improvement to the above in that already updated values are used in the iteration. For example, if the the nodes are reached by sweeping along i for each j and then along j , values at nodes $(i-1, j-1)$, $(i, j-1)$, $(i+1, j-1)$ and $(i-1, j)$ would have been updated prior to that at node (i, j) . With the recent values used, the iterative scheme can be written as

$$x_{i,j}^{m+1} = \frac{1}{2(A^m + C^m)} \left[A^m (x_{i+1,j}^m + x_{i-1,j}^{m+1}) - \frac{B^m}{2} (x_{i+1,j+1}^m + x_{i-1,j-1}^{m+1} - x_{i+1,j-1}^{m+1} - x_{i-1,j+1}^{m+1}) + C^m (x_{i,j+1}^m + x_{i,j-1}^{m+1}) \right]$$

The iteration is repeated until convergence to user-specified level.

Successive Over-Relaxation (SOR) Iteration Method. To suppress overshoots from iteration to iteration, one can also try relaxaation method. In this method, the iteration is carried out as

$$x_{i,j}^{m+1} + (\omega - 1)x_{i,j}^m = \frac{\omega}{2(A^m + C^m)} \left[A^m (x_{i+1,j}^m + x_{i-1,j}^m) - \frac{B^m}{2} (x_{i+1,j+1}^m + x_{i-1,j-1}^m - x_{i+1,j-1}^m - x_{i-1,j+1}^m) + C^m (x_{i,j+1}^m + x_{i,j-1}^m) \right]$$

where ω is a relaxation factor between 1 and 2. Determination of the optimal value of ω is not trivial for equations as complex as in the present case. The optimum ω value is usually obtained by trial and error. Interestingly, one can observe that the relaxation method is akin to solving the following unsteady equation

$$x_\tau = Ax_{\xi\xi} - 2Bx_{\xi\eta} + Cx_{\eta\eta}$$

in the “time” domain until steady state ($x_\tau = 0$) is reached! The iteration counter m now corresponds to the discrete time (ie., $\tau = m\delta\tau$) and the relaxation factor ω to the time-step size $\delta\tau$. One may refer to literature for more sophisticated iterative techniques for solving quasi-linear elliptic partial differential equations.