1. Analytical transformations
2. Algebraic grid generation
3. Elliptic grid generation
4. Variational grid generation
5. Hyperbolic grid generation

In general, the procedure to generate a grid goes as follows. First generate grids on the edges of the domain, second from the given edges, generate the sides, and finally generate the volume grid from the six given side grids. In the description below, we will assume two space dimensions, but the methods extends straightforwardly to three dimensions.

Analytical transformations. This is the most obvious type of grid generation. If the geometry is simple enough that we know an analytical transformation to the unit square, the grid generation becomes very easy. E.g., the domain between two circles,

$$D = \{(x, y) \mid r_i^2 \leq x^2 + y^2 \leq r_o^2\},$$

where r_i is the radius of the inner circle and r_o is the radius of the outer circle, can be mapped to the unit square by the mapping

$$\begin{aligned} x &= (r_i + (r_o - r_i)r)\cos 2\pi s \\ y &= (r_i + (r_o - r_i)r)\sin 2\pi s \end{aligned}$$

where now $0 \leq r \leq 1$ and $0 \leq s \leq 1$. The grid is then obtained by a uniform subdivision of the (r, s) coordinates, i.e., a grid with $m \times n$ grid points is given by

$$\begin{aligned} x_{i,j} &= x((i-1)/(m-1), (j-1)/(n-1)), \quad i = 1, 2, \ldots, m \quad j = 1, 2, \ldots, n \\ y_{i,j} &= y((i-1)/(m-1), (j-1)/(n-1)) \end{aligned}$$

This method of generating grids is simple and efficient. Another advantage is that we can generate orthogonal grids if a conformal mapping can be found. Orthogonal grids are grids where grid lines always intersect at straight angles. Often PDE approximations becomes more accurate, and have stencils with fewer points on orthogonal grids. Of course, this method has very limited generality, and can only be used for very special geometries.

Algebraic grid generation. This type of grid generation is also called transfinite interpolation. In its simplest form, it is described by the formula

$$\begin{aligned} x(r,s) = \;& (1-r)x(0,s) + rx(1,s) + (1-s)x(r,0) + sx(r,1) - \\ & (1-r)(1-s)x(0,0) - r(1-s)x(1,0) - (1-r)sx(0,1) - rsx(1,1) \\ y(r,s) = \;& (1-r)y(0,s) + ry(1,s) + (1-s)y(r,0) + sy(r,1) - \\ & (1-r)(1-s)y(0,0) - r(1-s)y(1,0) - (1-r)sy(0,1) - rsy(1,1) \end{aligned} \tag{2.2}$$

Here it is assumed that $(x(r,s), y(r,s))$ are known on the sides of the domain. Formula (2.2) gives then the grid in the entire domain. The formula is an interpolation from the sides to the interior. The two first terms is the interpolation between the sides $r = 0$ and $r = 1$, the next two terms are the same for the s direction. Finally four corner terms are subtracted. One can easily verify that (2.2) is exact on the boundary by putting $r = 0$, $r = 1$, $s = 0$, or $s = 1$.

This formula can be generalized to the case where

- x and y are given on several coordinate lines in the interior of the domain.
- both x and y and its derivatives are given on the sides.

A general form of (2.2) is obtained by introducing the blending functions $\varphi_0(r)$ and $\varphi_1(r)$, with the properties

$$\varphi_0(0) = 1 \quad \varphi_0(1) = 0 \quad \varphi_1(0) = 0 \quad \varphi_1(1) = 1$$

instead of the functions $1 - r$ and r in (2.2). The general transfinite interpolation is then defined by

$$\begin{aligned} x(r,s) = \quad & \varphi_0(r)x(0,s) + \varphi_1(r)x(1,s) + \varphi_0(s)x(r,0) + \varphi_1(s)x(r,1) - \\ & \varphi_0(r)\varphi_0(s)x(0,0) - \varphi_1(r)\varphi_0(s)x(1,0) - \varphi_0(r)\varphi_1(s)x(0,1) - \varphi_1(r)\varphi_1(s)x(1,1) \end{aligned}$$

By introducing the projector

$$P_r(x)(r,s) = \varphi_0(r)x(0,s) + \varphi_1(r)x(1,s)$$

we can write the transfinite interpolation above as the boolean sum

$$x = P_r(x) + P_s(x) - P_r(P_s(x)) \tag{2.3}$$

Note that $P_r(x)$ denotes a function of (r,s). The (r,s) dependency was not written out in formula (2.3). Of course, it is possible to use different projectors and blending functions in the different coordinate directions. The projector form (2.3) is convenient for describing generalizations of the method, e.g., if the derivatives of the grid are prescribed on the boundary, we can use the projector

$$P_r(x) = \varphi_0(r)x(0,s) + \varphi_1(r)x(1,s) + \psi_0(r)\frac{\partial x(0,s)}{\partial r} + \psi_1(r)\frac{\partial x(1,s)}{\partial r} \tag{2.4}$$

in (2.3) instead. Here the new blending function ψ_0 satisfies

$$\begin{aligned} \psi_0(0) = 0 \quad \psi_0(1) = 0 \quad \psi_0'(0) = 1 \quad \psi_0'(1) = 0 \\ \psi_1(0) = 0 \quad \psi_1(1) = 0 \quad \psi_1'(0) = 0 \quad \psi_1'(1) = 1 \end{aligned}$$

The functions $\varphi_0(r)$ and $\varphi_1(r)$ satisfy the same conditions as previously plus the additional conditions

$$\varphi_0'(0) = 0 \quad \varphi_0'(1) = 0 \quad \varphi_1'(0) = 0 \quad \varphi_1'(1) = 0$$

When the blending functions satisfy all the above conditions, it is not hard to verify that

$$\begin{aligned} P_r(x)(0,s) &= x(0,s) \quad P_r(x)(1,s) = x(1,s) \\ \tfrac{\partial P_r(x)}{\partial r}(0,s) &= \tfrac{\partial x}{\partial r}(0,s) \quad \tfrac{\partial P_r(x)}{\partial r}(1,s) = \tfrac{\partial x}{\partial r}(1,s) \end{aligned}$$

An example of blending functions satisfying all the conditions is

$$\begin{aligned} \varphi_0(r) &= (2r+1)(1-r)^2 & \varphi_1(r) &= (3-2r)r^2 \\ \psi_0(r) &= (1-r)^2 r & \psi_1(r) &= (r-1)r^2 \end{aligned}$$

An example of when we would like to prescribe the derivatives on the boundary, is when we want the grid to be orthogonal to the boundary. This can sometimes be needed in order to simplify boundary conditions.

Say for example, that we would like the grid to be orthogonal to the boundary given by $r = 0$, $(x(0,s), y(0,s))$. The vector $(\frac{\partial x}{\partial s} \ \frac{\partial y}{\partial s})$ is tangential to the boundary, and the vector $(\frac{\partial x}{\partial r} \ \frac{\partial y}{\partial r})$ is tangential to the r-coordinate line. The orthogonality conditions therefore is

$$\frac{\partial x}{\partial s}\frac{\partial x}{\partial r} + \frac{\partial y}{\partial s}\frac{\partial y}{\partial r} = 0 \tag{2.5}$$

at the boundary points $(0,s)$. The s-derivatives can be evaluated directly from the boundary description $(x(0,s), y(0,s)$. In order to determine the boundary r-derivatives, we need an additional equation. We can for example specify the distance between the first and second grid lines away from the boundary. For many applications, it is desirable to make such a specification of the grid resolution near the boundary. Denote the distance between the first grid lines in the r-direction $d(s)$. The relationship defining $d(s)$ is

$$(x(\Delta r, s) - x(0,s))^2 + (y(\Delta r, s) - y(0,s))^2 = d(s)^2.$$

With the approximation $x(\Delta r, s) - x(0,s) \approx \partial x/\partial r \Delta r$, we obtain

$$\left(\frac{\partial x}{\partial r}\right)^2 + \left(\frac{\partial y}{\partial r}\right)^2 = \frac{d(s)^2}{\Delta r^2} \tag{2.6}$$

at the boundary points $(0,s)$. We can now solve the two equations (2.5) and (2.6) for the two r-derivatives at the boundary. The result is

$$\frac{\partial x}{\partial r} = \frac{d(s)}{\Delta r}\frac{\frac{\partial y}{\partial s}}{\sqrt{\left(\frac{\partial x}{\partial s}\right)^2 + \left(\frac{\partial y}{\partial s}\right)^2}}$$

and

$$\frac{\partial y}{\partial r} = -\frac{d(s)}{\Delta r}\frac{\frac{\partial x}{\partial s}}{\sqrt{\left(\frac{\partial x}{\partial s}\right)^2 + \left(\frac{\partial y}{\partial s}\right)^2}}.$$

These formulas are used at the boundary to compute the r-derivatives of x and y there. Then these r-derivatives are inserted into the formula (2.4) and the grid is generated using this projector in the r-direction.

The specification of $d(s)/\Delta r$ is often made as $d(s) = \alpha \Delta r$, where α is a factor which can be interpreted as a percentage of the spacing near the boundary relative an ideal uniform spacing.

The advantage of algebraic grid generation is its efficiency, and ease of implementation. A disadvantage is that there is no guarantee that the method will be successful. For very curved boundaries, it can often happen that grid lines intersect, like the example in Figure 1.8 a). Another problem is the propagation of singularities from the boundary. If the boundary has an interior corner, as in Figure 1.8 b), the corner will be seen in the interior domain, making it difficult to generate smooth grids.

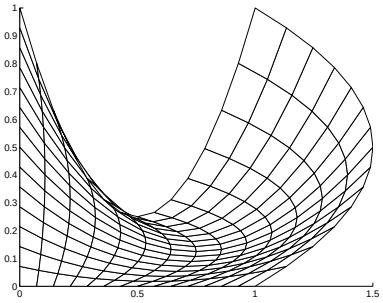


FIG. 1.8a. *Folding of grid lines.*

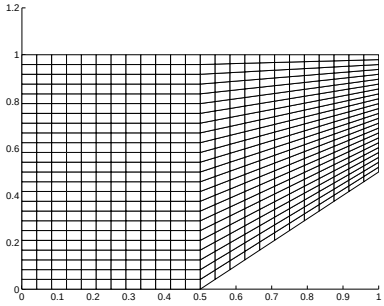


FIG. 1.8b. *Propagating corner.*

Elliptic grid generation. This type of grid generation is motivated by the maximum principle for elliptic PDEs. We define the inverse grid transformation, $r(x, y), s(x, y)$ as the solution of

$$\begin{aligned} r_{xx} + r_{yy} &= 0 \\ s_{xx} + s_{yy} &= 0 \end{aligned} \tag{2.7}$$

We here use the notation

$$x_r = \frac{\partial x}{\partial r}, \quad x_{rr} = \frac{\partial^2 x}{\partial r^2} x_s = \frac{\partial x}{\partial s}, \quad x_{ss} = \frac{\partial^2 x}{\partial s^2}$$

We know that $0 \leq r \leq 1$ and $0 \leq s \leq 1$ are monotone on the boundaries. It then follows from the maximum principle that r and s will stay between these values. Furthermore, there will be no local extrema in the interior, and thus grid lines can not fold. The equations (2.7) are formulated in the x-y domain, and has to be transformed to the unit square, so that we can solve them there. We use the unknown transformation itself to transform the equations (2.7). The transformed system then becomes

$$\begin{aligned} (x_s^2 + y_s^2)x_{rr} - 2(x_r x_s + y_r y_s)x_{rs} + (x_r^2 + y_r^2)x_{ss} &= 0 \\ (x_s^2 + y_s^2)y_{rr} - 2(x_r x_s + y_r y_s)y_{rs} + (x_r^2 + y_r^2)y_{ss} &= 0 \end{aligned} .$$

This problem can be solved as a Dirichlet problem if (x, y) are given on the boundaries, or as a Neumann problem if the normal derivatives of (x, y) are specified on the boundaries. Specifying normal derivatives is here equivalent to specifying the distance between the first and second grid lines. These equations are then approximated by, e.g.,

$$\begin{aligned} x_r &\approx (x_{i+1,j} - x_{i-1,j})/2 \quad x_s \approx (x_{i,j+1} - x_{i,j-1})/2 \\ x_{rr} &\approx x_{i+1,j} - 2x_{i,j} + x_{i-1,j} \quad \text{etc.} \end{aligned}$$

where now the index space $1 \leq i \leq m$ and $1 \leq j \leq n$ is a uniform subdivision of the (r, s) coordinates, $r = (i-1)/(m-1)$, $s = (j-1)/(n-1)$. The number of grid points is specified as $m \times n$.

```
#include <math.h>

/* One iteration of Gauss-Seidel iteration for elliptic grid generator */

void elliptic( int m, int n, double* x, double* y, double* err )
{
  int i, j, ind;
  double xtemp, ytemp, g11, g12, g22;

  *err = 0;
  for( j=1 ; j<n-1 ; j++ )
    for( i=1 ; i<m-1 ; i++ )
    {
      ind = i + m*j;
      g11 =( (x[ind+1]-x[ind-1])*(x[ind+1]-x[ind-1]) +
             (y[ind+1]-y[ind-1])*(y[ind+1]-y[ind-1]) )/4;
      g22 =( (x[ind+m]-x[ind-m])*(x[ind+m]-x[ind-m]) +
             (y[ind+m]-y[ind-m])*(y[ind+m]-y[ind-m]) )/4;
      g12 =( (x[ind+1]-x[ind-1])*(x[ind+m]-x[ind-m]) +
             (y[ind+1]-y[ind-1])*(y[ind+m]-y[ind-m]) )/4;

      xtemp = 1/(2*(g11+g22))*(
         g22*x[ind+1] - 0.5*g12*x[ind+1+m] + 0.5*g12*x[ind+1-m] +
         g11*x[ind+m] + g11*x[ind-m] +
         g22*x[ind-1] - 0.5*g12*x[ind-1-m] + 0.5*g12*x[ind-1+m] );

      ytemp = 1/(2*(g11+g22))*(
         g22*y[ind+1] - 0.5*g12*y[ind+1+m] + 0.5*g12*y[ind+1-m] +
         g11*y[ind+m] + g11*y[ind-m] +
         g22*y[ind-1] - 0.5*g12*y[ind-1-m] + 0.5*g12*y[ind-1+m] );

      *err += (x[ind]-xtemp)*(x[ind]-xtemp)+(y[ind]-ytemp)*(y[ind]-ytemp);

      x[ind] = xtemp;
      y[ind] = ytemp;
    }
  *err = sqrt( *err/((m-2)*(n-2)) );
}
```

CODE 1.2. *Gauss-Seidel iteration for elliptic grid generator.*

The equations can then be solved by a standard elliptic solver such as, e.g., conjugate gradients, Gauss-Seidel or the multi grid method. In Code 1.2, we show a C function for doing one Gauss-Seidel iteration on the system. The index convention of Code 1.1, using one dimensional arrays to store matrices is used here too.

If you want to specify both the grid and its normal derivatives on the boundary, the second order elliptic PDE above can not be used, but it is possible to define an elliptic problem with fourth order derivatives instead.

Elliptic grid generation is very reliable, and will always produce a grid. However it might not always be the grid you want. For example, grid lines tend to cluster near convex boundaries, but will be very sparsely distributed near concave boundaries. In Figures 1.9a and 1.9b we show the same example as in Figures 1.8a and 1.8b, but now the grid is generated by the elliptic equations (2.7). Clearly, the problems which occured with transfinite interpolation does not happen here.

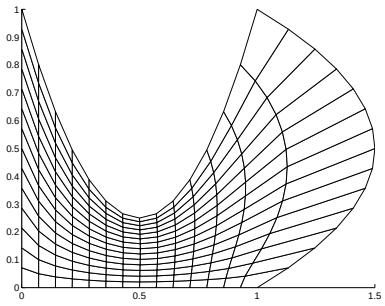


FIG. 1.9a. *No folding.*

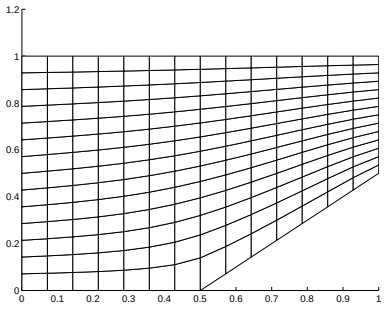


FIG. 1.9b. *Corner smoothed.*

To introduce more control over the grid, so called control functions are introduced into (2.7). The system then becomes

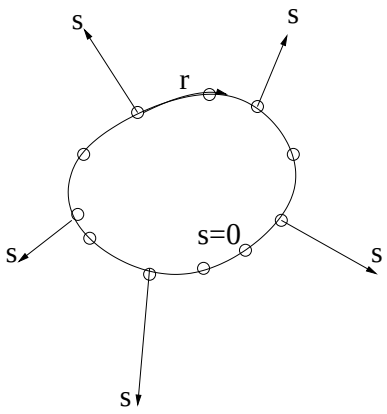
$$r_{xx} + r_{yy} = P$$
$$s_{xx} + s_{yy} = Q$$

Alternatively the functions (P, Q) can be scaled so that the right hand side becomes $(r_x^2 + r_y^2)P$ and $(s_x^2 + s_y^2)Q$. The maximum principle is now lost, so that we have no guarantee that a grid will be successfully generated. There is a risk that grids with folded coordinate lines occur.

The functions P and Q can be chosen to attract grid lines to certain lines or points in the space. To do this by hand is very difficult. Often P and Q are specified from a weight function, which specifies the grid density at each point in the domain. This is often used in adaptive methods, when the weight function measures the error in a computed solution.

Elliptic grid generators are often used to post process grids computed by algebraic grid generators. If (2.3) is solved by an iterative method for elliptic PDEs, it is sufficient to do a few iterations to smooth out irregularities in the grid. We do not need to solve the equations (2.3) themselves to any high accuracy, thereby saving computational power.

Hyperbolic grid generation. Hyperbolic grid generation is suitable for generating grids external to an object. The method takes the boundary of the object as initial data, and generates the grid by marching in the direction normal to and out from the object. The figure below gives an idea of how the problem is set up.



Grid points at inner boundary, s directed out from object.

The PDE to be solved is

$$\begin{aligned} \frac{\partial x}{\partial r}\frac{\partial x}{\partial s} + \frac{\partial y}{\partial r}\frac{\partial y}{\partial s} &= 0 \\ \frac{\partial x}{\partial r}\frac{\partial y}{\partial s} - \frac{\partial x}{\partial s}\frac{\partial y}{\partial r} &= v(r,s) \end{aligned} \tag{2.8}$$

where $v(r,s)$ is a function given by the user. The first equations states that the grid is orthogonal, and the second equation specifies the determinant of the grid mapping should be equal to $v(r,s)$. The meaning of $v(r,s)$ is that it is proportional to the area of the grid cells. For example the cell with corners (i,j), $(i+1,j)$, $(i,j+1)$, and $(i+1,j+1)$ has area

$$A_{i,j} = (x_{i+1,j} - x_{i,j})(y_{i,j+1} - y_{i,j}) - (x_{i,j+1} - x_{i,j})(y_{i,+1j} - y_{i,j})$$

By approximating $x_{i+1,j} - x_{i,j} \approx \partial x/\partial r_{i,j}\Delta r$, we obtain that

$$A_{i,j} \approx (\frac{\partial x}{\partial r}\frac{\partial y}{\partial s} - \frac{\partial x}{\partial s}\frac{\partial y}{\partial r})\Delta r\Delta s = v(r_i, s_j)\Delta r\Delta s$$

so that taking $v(r,s) = 1$ gives each cell the same area as that of an imagined uniform grid.

We solve (2.8) as a linear system for the s-derivatives, and obtain

$$\begin{pmatrix} \frac{\partial x}{\partial s} \\ \frac{\partial y}{\partial s} \end{pmatrix} = \frac{v(r,s)}{\left(\frac{\partial x}{\partial r}\right)^2 + \left(\frac{\partial y}{\partial r}\right)^2} \begin{pmatrix} -\frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial r} \end{pmatrix} \tag{2.9}$$

This can be interpreted as an initial value problem, where x and y are given functions of r for $s = 0$ (the inner boundary). The grid is generated by solving the system for $s > 0$, so that s has the same function as time in a time dependent problem. In order to verify that the system is hyperbolic, and thus well-posed so that it can be solved, we should write it on the form

$$\begin{pmatrix} \frac{\partial x}{\partial s} \\ \frac{\partial y}{\partial s} \end{pmatrix} = A \begin{pmatrix} \frac{\partial x}{\partial r} \\ \frac{\partial y}{\partial r} \end{pmatrix}$$

where A is a 2×2 matrix, and verify that the eigenvalues of A are real and distinct (or that a complete basis of eigenvectors exist). However, (2.9) is non-linear and can not be written on the standard form. We instead consider the linearized problem, i.e., assume that $\hat{x}, \hat{y}$ is a solution, and study the solution $\hat{x} + \tilde{x}, \hat{y} + \tilde{y}$ where $\tilde{x}$ and $\tilde{y}$ are small perturbations. Inserting this into (2.9), and neglecting all powers of two and higher of $\tilde{x}$ and $\tilde{y}$ gives the linearized system

$$\begin{pmatrix} \frac{\partial \tilde{x}}{\partial s} \\ \frac{\partial \tilde{y}}{\partial s} \end{pmatrix} = A \begin{pmatrix} \frac{\partial \tilde{x}}{\partial r} \\ \frac{\partial \tilde{y}}{\partial r} \end{pmatrix}$$

where A is given by

$$\frac{v(r,s)}{(\hat{x}_r^2 + \hat{y}_r^2)^2} \begin{pmatrix} 2\hat{x}_r\hat{y}_r & \hat{y}_r^2 - \hat{x}_r^2 \\ \hat{y}_r^2 - \hat{x}_r^2 & -2\hat{x}_r\hat{y}_r \end{pmatrix}$$

where the notation $x_r = \partial x/\partial r$, and similarly for y and s is used. We note that A is symmetric. The eigenvalues are given by

$$\pm\frac{v(r,s)}{\hat{x}_r^2 + \hat{y}_r^2}.$$

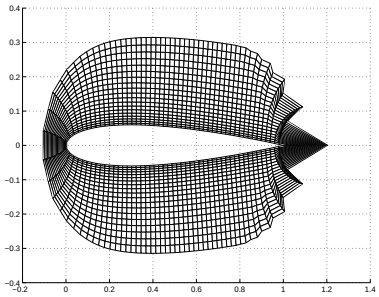
We have thus verified that the linearized system is hyperbolic.

The implementation of the hyperbolic system, is conveniently done by first writing it on semi-discrite form,

$$\begin{pmatrix} \frac{dx_i(s)}{ds} \\ \frac{dy_i(s)}{ds} \end{pmatrix} = \frac{v(r_i,s)}{\left(\frac{x_{i+1}(s)-x_{i-1}(s)}{2\Delta r}\right)^2 + \left(\frac{y_{i+1}(s)-y_{i-1}(s)}{2\Delta r}\right)^2} \begin{pmatrix} -\frac{y_{i+1}(s)-y_{i-1}(s)}{2\Delta r} \\ \frac{x_{i+1}(s)-x_{i-1}(s)}{2\Delta r} \end{pmatrix} \tag{2.10}$$

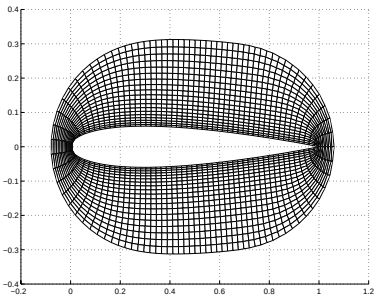
The system is now an ODE in the s variable, which can be solved by a suitable method for initial value problems, for example, a Runge-Kutta method. An artificial dissipation term could be added to the right hand side, in order to improve numerical stability. Of course the second order accurate centered difference method used here could, if desired, be replaced by any other approximation of the r-derivatives.

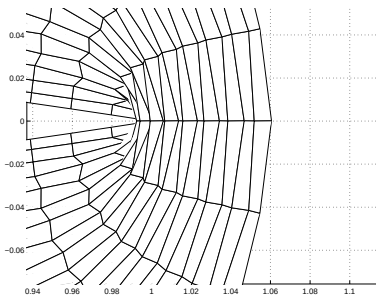
Some examples for the construction of a grid around a airplane wing profile are shown below. The first picture was obtained by integrating (2.10) using the forward Euler method, which gives an unstable difference scheme. The effect is clearly seen.



Unstable integration method, gives failed grid

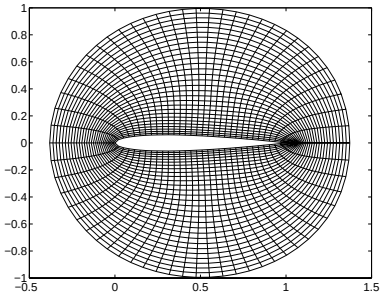
Next a difference scheme with very poor accuracy (the Lax-Friedrichs scheme) was used to solve (2.10). The result is shown below. There is too much numerical dissipation in the scheme, which causes the grid to fold, as seen in the close up.



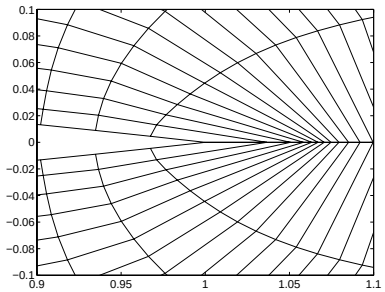


a. *High numerical viscosity, too much smoothing.* b. *Close up of trailing edge region.*

Finally, the result from using the fourth order accurate Runge-Kutta method in (2.10) for the s integration is shown below. The grid is now without foldings, and could be used for a computation. There is a clustering of points near the trailing edge, as seen in the close up, but this could be improved by better choise of the function $v(r,s)$. In all examples it was taken equal to $v = 0.5 + 2.5s$, so that the stretching factor goes from 0.5 near the wall to 3 far away.



a. *Runge-Kutta 4th order, accurate solution.*



b. *Close up of trailing edge region.*

Hyperbolic grid generation has the advantage that an orthogonal grid is generated if the algorithm is successful. The computational cost is considerably lower than for the elliptic grid generator. Only the interior boundary can be specified. This can be a disadvantage if the user has requirements on where the outer boundary of the final grid should be. Another disadvantage is that singularities such as discontinuities can form in the solution of hyperbolic systems. When singularites form, the grid generation fails. Furthermore, when solving (2.10) by explicit s-stepping, failure can also occur when numerical stability constraints, e.g., the CFL-condition, are not satisfied.

Variational methods. These methods have evolved from elliptic grid generation. To solve an elliptic PDE is often equivalent to minimizing a functional. We distinguish three functionals which describe the grid quality. The orthogonality functional

$$O = O_1 + O_2 + O_3 + O_4,$$

each of the four components O_i is related to the angle between grid lines. See Figure 1.10. We have for the first angle

$$O_1 = \sum_{i=1,j=1}^{n_i-1,n_j-1} ((\mathbf{r}_{i,j+1} - \mathbf{r}_{i,j})^T(\mathbf{r}_{i+1,j} - \mathbf{r}_{i,j}))^2$$

where $\mathbf{r} = (x\ y)^T$. Here we let $\mathbf{a}^T$ denote the transposed vector, and $\mathbf{a}^T\mathbf{b}$ becomes then the scalar product between $\mathbf{a}$ and $\mathbf{b}$. O is zero for an orthogonal grid.

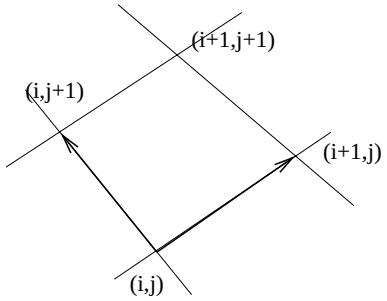


FIG. 1.10 *Scalar product between intersecting grid lines.*

The spacing functional measures how fast the spacing changes, and is defined by

$$S = S_H + S_V.$$

Here S_H and S_V are horizontal and vertical spacing respectively. The definition of S_H is

$$S_H = \sum_{i=1}^{n_i-1} \sum_{j=1}^{n_j} |\mathbf{r}_{i+1,j} - \mathbf{r}_{i,j}|^2$$

and similarly for S_V. S is minimal for a uniform grid. The cell area functional is defined by,

$$A = \sum_{i=1,j=1}^{n_i-1,n_j-1} A_{i,j}^2,$$

and measures how much the cell area changes over the grid. $A_{i,j}$ is defined as the area of the cell with corners $(i,j),(i+1,j),(i,j+1),(i+1,j+1)$.

These three functionals are combined into one, which is minimized. We obtain the problem to minimize

$$V = aA + bS + cO$$

here a, b and c are parameters, which the user should provide. They indicate the relative importance of the different quality measures are for the present problem. To, e.g., have a grid which is close to orthogonal, c should be big, and a and b small.

The minimization can be done by a suitable numerical method, such as, e.g., the conjugate gradient method.

The advantage of this method is flexibility. However, we have no guarantee for success. E.g., minimizing the spacing functional corresponds to solving the elliptic PDE

$$x_{rr} + x_{ss} = 0$$

$$y_{rr} + y_{ss} = 0$$

which, since formulated in physical space, is not the same as (2.7). Here we have no help from the maximum principle.

Remark. It can be advantageous to mix grid generation methods. E.g., one could use an elliptic method to generate grids on the sides of the domain, and then an algebraic method to generate the grid in the entire volume. This gives good performance, since algebraic grid generation is inexpensive compared with elliptic grid generation.

2.4 Implementational considerations

It is convenient to describe all boundary curves in a compatible parameterization. It is a good idea to use the normalized arc length as curve parameter. To see why, consider the airfoil (NACA0012) in Figure 1.11 below. It is described by the equation

$$y = a_0\sqrt{x} + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \quad 0 \le x \le 1$$

The constants a_i have given values. The most straightforward way to place points on the airfoil is to make a uniform discretization of the x-axis, $x_i = i\Delta x, i = 1,\ldots,N$, and use the points $(x_i, y(x_i))$ as boundary grid points. However, we would then obtain the picture in Figure 1.12a. There are very few points at the sharp gradient in the front. This

local sparseness influences the entire grid when later the points on the airfoil are used to generate the grid in the entire domain. Of course stretching functions can be used to cluster more points near the leading edge, but it is difficult to tune the right stretching by hand.

We instead write the airfoil as a curve, $(x(s), y(s))$, where $s \in [0, 1]$ is proportional to the arc length. This is done by the reparameterization

$$s(p) = \frac{1}{L} \int_a^p \sqrt{x'(t)^2 + y'(t)^2}\, dt \tag{2.11}$$

Here L is the total arc length, $L = \int_a^b \sqrt{x'(t)^2 + y'(t)^2}\, dt$. The function $x(s)$ is implemented according to the following algorithm. Given a value of s, we solve equation (2.11) for p, using Newton's method, and numerical quadrature in the integral. The x value to return is then $x(p)$, which we easily compute from the original parameterization.

Using the same number of points as in Figure 1.12a, but with the arc length as parameter, we obtain the picture in Figure 1.12b. There is clearly an improvement.

Another problem that has to be dealt with, is the coordinate direction of opposite sides. Figure 1.13 illustrates this. The arc length coordinate must run in the same direction on opposite sides. Every grid generating software must be capable of keeping track of the coordinate directions on facing sides, and, if necessary, apply the transformation $s \to 1-s$. This problem becomes more severe in three space dimensions.

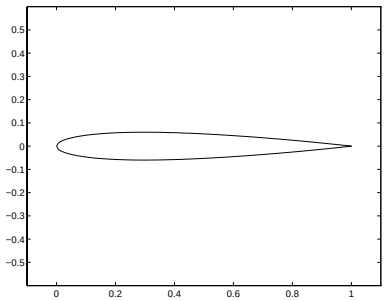


FIG. 1.11 *NACA0012 airfoil.*

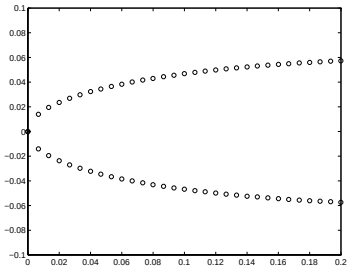


FIG. 1.12a. *x as parameter.*

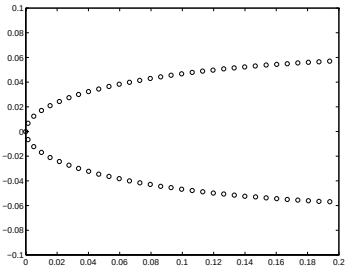


FIG. 1.12b. *Arc length as parameter.*

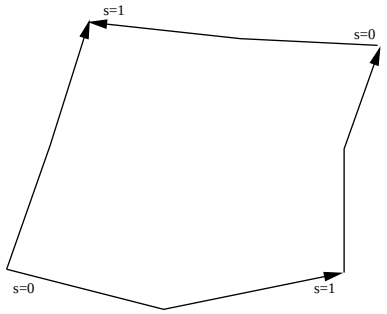


FIG. 1.13 *Coordinate directions on upper and lower sides do not match.*

2.5 Stretching transformations

It is often necessary to cluster points more densely near certain critical parts of the computational domain. This can be achieved by a so called stretching function. This is a function which maps the unit interval onto itself. We denote the stretching function by $s(u) : [0,1] \to [0,1]$. Stretching is applied in only one coordinate direction. It is of course possible to define different stretching functions in different coordinate directions. Along one coordinate, we go through following steps in order to generate grid points on one coordinate line.

1. Take a uniform grid, $u_i = (i-1)/(n-1), i = 1, \ldots, n$
2. Apply stretching $s_i = s(u_i)$
3. Generate points in the arc length parameter, i.e., use the points $(x(s_i), y(s_i), z(s_i))$ on one edge.

The philosophy behind this is that grids should be stretched as a separate first step before the grid generation. This is in contrast with the discussion on elliptic grid generators, one takes there the point of view that all grid stretchings should be build into the control functions P, Q.

We next give some examples of suitable grid stretching functions. Assume that we have the grid points $x_1 = 0, x_2, x_3, \ldots, x_n = 1$. A common practise for the computation of boundary layers in fluid dynamics is to use the following spacing

$$x_{i+1} - x_i = \alpha(x_i - x_{i-1}) \tag{2.12}$$

with $\alpha > 1$. Here we assume that the layer is near the boundary $x_1 = 0$. We specify the smallest spacing $x_2 - x_1$, and then gradually increase the spacing with i through (2.12). We have $x_{i+1} - x_i = \alpha^{i-1}(x_2 - x_1)$, and thus

$$x_i = x_{i-1} + \alpha^{i-2}(x_2 - x_1) = \ldots = x_1 + (1 + \alpha + \ldots + \alpha^{i-2})(x_2 - x_1) = \frac{\alpha^{i-1} - 1}{\alpha - 1}(x_2 - x_1)$$

Because $x_n = 1$, we obtain $x_2 - x_1 = (\alpha - 1)/(\alpha^{n-1} - 1)$, and thus

$$x_i = \frac{\alpha^{i-1} - 1}{\alpha^{n-1} - 1}$$

Let the uniform spacing variable be $u = (i-1)/(n-1)$. The stretching function associated with (2.12) is then

$$x(u) = \frac{\alpha^{(n-1)u} - 1}{\alpha^{n-1} - 1}$$

This function must be well defined for all n, which implies that $\alpha = 1 + \mathcal{O}(1/n)$. Since $\alpha^{n-1} \to e^{\beta}$ as $n \to \infty$, we prefer to define the stretching as

$$x(u) = \frac{e^{\beta u} - 1}{e^{\beta} - 1}$$

where the parameter $\beta > 0$ is chosen to control the strength of the stretching. Often the derivative $x'(0)$ is specified as a measure of the stretching near the boundary. Beta can then easily be computed from $x'(0)$.

The exponential stretching is very simple. Analysis of truncation errors have led to more advanced stretching functions, such as the hyperbolic tangent,

$$x(u) = 1 + \frac{\tanh \delta(u-1)/2}{\tanh \delta/2}. \tag{2.13}$$

The parameter δ is used to control the strength of the stretching, and can be determined from a specified value of $x'(0)$. Examples of grid point distributions are given below.

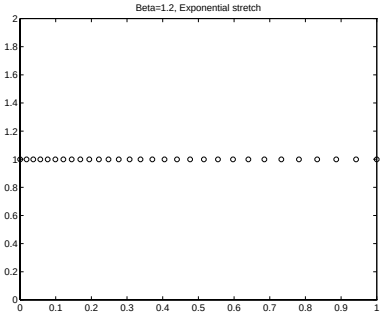


FIG. 1.14a. $\beta = 1.2$.

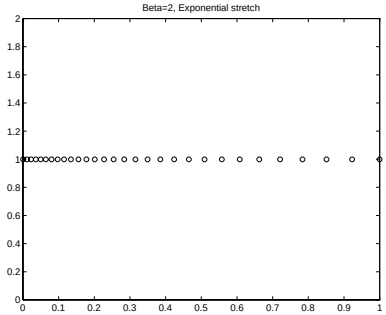


FIG. 1.14b. $\beta = 2$.

Exponential stretching.

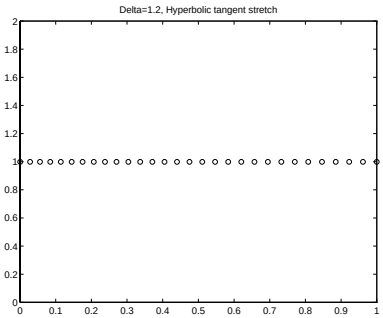


FIG. 1.15a. $\delta = 1.2$.

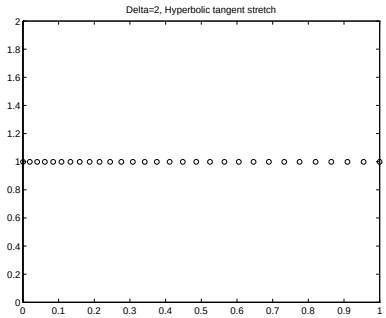


FIG. 1.15b. $\delta = 2$.

Hyperbolic tangent stretching.

More general forms of the functions above can be derived to place the stretching at an arbitrary point, not just near the left boundary. An example of the hyperbolic tangent stretching function is then

$$x(u) = \frac{\tanh(b(u-c)) - A}{B - A}$$

where $A = \tanh(b(-c))$ and $B = \tanh(b(1-c))$. Here b is the strength and $1-c$ the location of the attraction point. The formula above becomes (2.13) under the restriction $c = 1$, (with $b = \delta/2$).

2.6 Further reading

Some references on grid generation are:

J. Thompson, Z. Warsi, and C. Mastin, *Numerical Grid Generation*, North-Holland, 1985, ISBN 0-444-00985-X.

P. Knupp and S. Steinberg, *Fundamentals of Grid Generation*, CRC Press, 1993, ISBN 0-8493-8987-9.

Thompson, Soni, Weatherill, *Handbook of Grid Generation*, CRC Press, 1999, ISBN 0-8493-2687-7.