Note : Examples and derivations are from chapter 9 of textbook Computational Fluid Dynamics Vol-I by Hoffmann and Chiang

Grid Generation-Structured Grids

In order to numerically solve the governing PDEs of fluid mechanics, approximations to the partial differentials are introduced. These approximations convert the partial derivatives to finite difference expressions, which are used to rewrite the PDEs as algebraic equations. The approximate algebraic equations, referred to as finite difference equations (FDEs) are eventually solved at discrete points within the domain of interest. Therefore, a set of grid points within the domain, as well as the boundaries of the domain, must be specified.

Typically, the computational domain is selected to be rectangular in shape where the interior grid points are distributed along grid lines. Therefore, the grid points can be identified easily with reference to the appropriate grid lines. This type of grid is known as the structured grid. A second category of grid system may be constructed where the grid points cannot be associated with orderly defined grid lines. This type of grid system is known as the unstructured grid. In this section, we discuss about structured grids. So, wherever grid generation is referred, structured grid generation is implied. In a rectangular physical domain, the generation of a grid, with uniform spacing is simple and easy. Grid points may be specified as coincident with the boundaries of the physical domain, thus making specification of boundary conditions considerably less complex. But the majority of the physical domains of interest are nonrectangular. Therefore, imposing a rectangular computational domain on such a physical domain will require some sort of interpolation for the implementation of the boundary conditions. Since the boundary conditions have a dominant influence on the solution of the equation, such an interpolation causes inaccuracies at the place of greatest insensitivity. In addition, unequal grid spacing near the boundaries creates further complications with the FDEs since approximations with nonequal stepsizes must be used. The form of the FDE changes from node to node, creating cumbersome programming details.

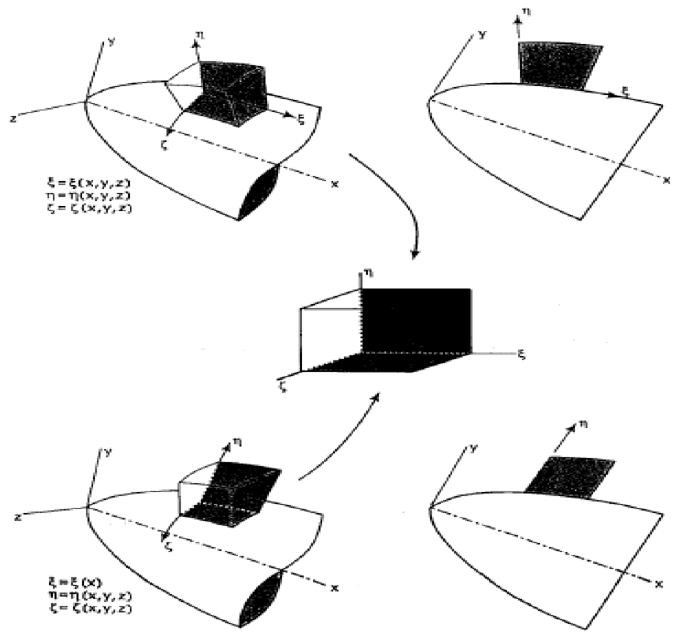


Figure 9.1. Nomenclature for the generalized coordinate system

To overcome, these difficulties, a transformation from physical space to computational space is introduced. This transformation is accomplished by specifying a generalized coordinate system which will map the nonrectangular grid system in the physical space to a rectangular uniform grid spacing in the computational space. The generalized coordinate system may be expressed in many ways. Two examples are illustrated in Figure 9.1. The first example shows the so-called body-fitted coordinate system where two coordinates lines, ξ and ζ, are aligned on the surface along the streamwise and circumferential directions, and where the third coordinate, η, is normal to the surface. In the second example, the ξ coordinate is aligned along the body axis, ζ is in the circumferential direction and η is normal to the body axis.

For illustrative purposes, 2D problems is considered in detail, with a description of 3D problems to follow. A 2D domain is illustrated in Figure 9.2.

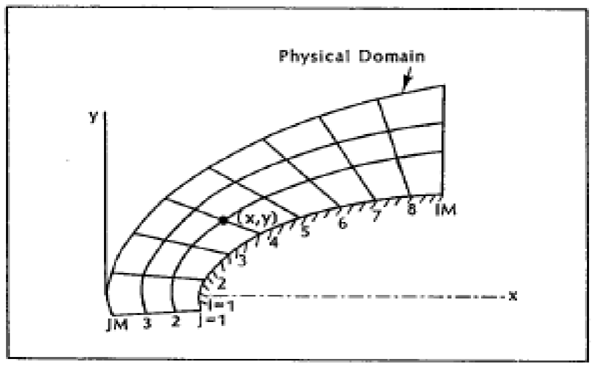


Figure 9.2. A typical 2D domain for the axisymmetric blunt body problem

In order to eliminate the difficulty associated with the nonequal stepsizes used in the FDEs, the physical domain is transformed into a rectangular, constant step-size, computational domain. A typical computational domain is shown in Figure 9.3. Note that the computational domain is obtained by deformation of the physical domain, i.e., by twisting and stretching , etc.

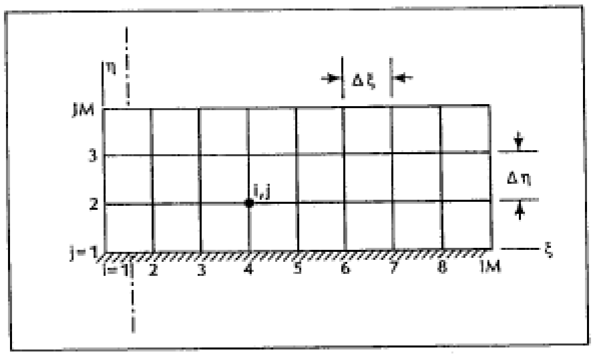


Figure 9.3. Computational domain with constant stepsizes $\Delta\xi$ and $\Delta\eta$

The central issue is identifying the location of the grid points in the physical domain i.e., *x* and *y* coordinates of a grid point in the physical space that corresponds to a grid point (*i, j*) specified in the computational domain. In determining the grid points, a few constraints must be imposed. First, the mapping must be one-to-one; i.e., grid lines of the same family cannot cross each other. Second, from a numerical point of view, a smooth grid point distribution with minimum grid line skewness, orthogonality or near orthogonality, and a concentration of grid points in regions where high flow gradient occur are required. Obviously, all of the desired features may not be met by the use of a particular grid generation technique. Grid generation techniques which emphasize any one or a combination of these features are described in the subsequent sections.

Generally, grid generation techniques may be classified as (1) algebraic methods, (2) particular differential methods, or (3) conformal mappings based on complex variables. In addition, the grid system may be categorized as fixed or adaptive. Obviously, a fixed grid system is generated prior to the solution of the governing equations of fluid motion and remains fixed independent of the solution. On the other hand, an adaptive grid system evolves as a result of the solution of the equations of fluid motion. For example, grid points may move towards regions of high gradients such as in the neighborhood shock wave.

Conformal mappings based on complex variables are limited to 2D problems and require a reasonable knowledge of complex variables. In addition, the determination of the mapping function is sometimes a difficult task. In the following sections, techniques based on algebraic and PDE methods are presented.

Transformation of the Governing Partial Differential Equations

The equations of fluid motion include the continuity, momentum and energy equations. In this section, a simple 2D problem is proposed to familiarize with the processes involved in the transformation of a PDE and the complexity of the resultant equation. It should be mentioned that the form and type of the transformed equation remains the same as the original PDE;i.e., if the original equation is parabolic, then the transformed equation is also parabolic. Now, define the following relations between the physical and computational spaces:

$$\xi = \xi(x, y) \tag{9.1}$$

$$\eta = \eta(x, y) \tag{9.2}$$

The chain rule for partial differentiation yields the following expression:

$$\frac{\partial}{\partial x} = \frac{\partial \xi}{\partial x}\frac{\partial}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial}{\partial \eta} \tag{9.3}$$

The partial derivative will be denoted using the subscript notation, i.e., $\frac{\partial \xi}{\partial x} = \xi_x$. Hence,

$$\frac{\partial}{\partial x} = \xi_x \frac{\partial}{\partial \xi} + \eta_x \frac{\partial}{\partial \eta} \tag{9.4}$$

and similarly,

$$\frac{\partial}{\partial y} = \xi_y \frac{\partial}{\partial \xi} + \eta_y \frac{\partial}{\partial \eta} \tag{9.5}$$

Now consider a model PDE, such as

$$\frac{\partial u}{\partial x} + a\frac{\partial u}{\partial y} = 0 \tag{9.6}$$

This equation may be transformed from physical space to computational space using Equations (9.4) and (9.5). As a result,

$$\xi_x \frac{\partial u}{\partial \xi} + \eta_x \frac{\partial u}{\partial \eta} + a\left(\xi_y \frac{\partial u}{\partial \xi} + \eta_y \frac{\partial u}{\partial \eta}\right) = 0$$

which can be rearranged as

$$(\xi_x + a\xi_y)\frac{\partial u}{\partial \xi} + \eta_x \frac{\partial u}{\partial \eta} + a\left(\xi_y \frac{\partial u}{\partial \xi} + \eta_y \frac{\partial u}{\partial \eta} \right) = 0 \qquad (9.7)$$

This equation is the one that will be solved in computational domain. Also the transformation derivatives ξ_x, ξ_y, η_x and η_y must be determined from the functional relations (9.1) and (9.2). Comparing the original PDE (9.6) and the transformed equation in (9.7), it is obvious the transformed equation is more complicated than the original equation. Advantages gained by using generalized coordinates are counterbalanced by the resultant complexity of the PDE. However, the advantages outbalance the complexity of the transformed PDE.

Metrices and the Jacobian Transformation

The transformation derivatives such as ξ_x, ξ_y, η_x and η_y are defined as the metrics of transformation or simply as the metrics. The interpretation of the metrics is made by considering the following approximation.

$$\xi_x = \frac{\partial \xi}{\partial x} \cong \frac{\Delta \xi}{\Delta x}$$

This expression indicates that the metrics represent the ratio of arc lengths in the computational space to that of the physical space.

From equations (9.1) and (9.2) the following differential expressions are obtained.

$$d\xi = \xi_x dx + \xi_y dy \qquad (9.8)$$

$$d\eta = \eta_x dx + \eta_y dy \qquad (9.9)$$

which are written in a compact form as

$$\begin{bmatrix} d\xi \\ d\eta \end{bmatrix} = \begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} \qquad (9.10)$$

Reversing the role of independent variables, i.e.,

$$x = x(\xi, \eta)$$

$$y = y(\xi, \eta)$$

The following may be written as

$$dx = x_\xi d\xi + x_\eta d\eta \tag{9.11}$$

$$dy = y_\xi d\xi + y_\eta d\eta \tag{9.12}$$

In a compact form they are written as

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{bmatrix} \begin{bmatrix} d\xi \\ d\eta \end{bmatrix} \tag{9.13}$$

Comparing Equations (9.10) and (9.13), it can be concluded that

$$\begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix} = \begin{bmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{bmatrix}^{-1}$$

From which

$$\xi_x = J y_\eta \tag{9.14}$$

$$\xi_y = -J x_\eta \tag{9.15}$$

$$\eta_x = -J y_\xi \tag{9.16}$$

$$\eta_y = -J x_\xi \tag{9.17}$$

and
$$J = \frac{1}{x_\xi y_\eta - y_\xi x_\eta} \tag{9.18}$$

and is defined as the Jacobian of transformation.

The Jacobian , *J*, is interpreted as the ratio of the areas (volumes in 3D) in the computational space to that of the physical space.

Note that actual values of the metrics or the Jacobian could be negative. The values depend on the specification of physical and computational grid systems. For grid generation methods where analaytical expressions for the metrics can be written , they may be analytically evaluated or determined numerically by the use of finite difference expressions.

Grid Generation Techniques

A grid system with the following features is desired:

1. A mapping which guarantees one-to-one correspondence ensuring grid lines of the same family do not cross each other;
2. Smoothness of the grid point distribution;
3. Orthogonality or near orthogonality of the grid lines;
4. Options for grid point clustering;

Some of the features enumerated as in (2)-(4) above, may not be achievable with a particular grid generation technique.

Algebraic Grid Generation Techniques

The simplest grid generation technique is the algebraic method. The major advantage of this scheme is the speed with which a grid can be generated. An algebraic equation is used to relate the grid points in the computational domain to those of the physical domain. This objective is met by using an interpolation scheme between the specified boundary grid points to generate the interior grid points. Clearly, many algebraic equations can be introduced for this purpose. To illustrate the procedure, consider the simple physical domain depicted in Figure 9.4.

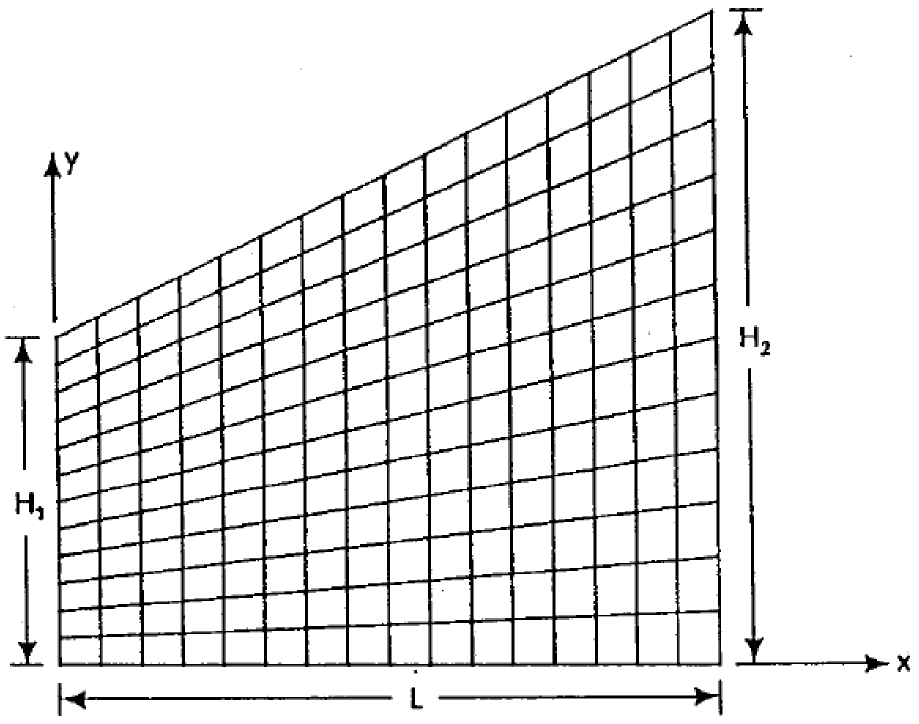


Figure 9.4. The physical space which must be transformed to a uniform rectangular computational space

Introducing the following algebraic relations will transform this nonrectangular physical domain into a rectangular domain:

$$\xi = x \tag{9.19}$$

$$\eta = \frac{y}{y_t} \tag{9.20}$$

In (9.20), y_t represents the upper boundary which is expressed as

$$y_t = H_1 + \frac{H_2 - H_1}{L} x$$

Thus, it may be written that

$$\xi = x \tag{9.21}$$

$$\eta = \frac{y}{H_1 + \frac{H_2 - H_1}{L}x} \tag{9.22}$$

Which can be arranged as

$$x = \xi \tag{9.23}$$

$$y = \left(H_1 + \frac{H_2 - H_1}{L}\xi\right)\eta \tag{9.24}$$

Pseduo code to find x and y interms of ξ and η

```
for i = 1:IM
    for j = 1:JM
        gsi(i,j) = (i-1)*dgsi;
        eta(i,j) = (j-1)*deta;
        x(i,j) = gsi(i,j);
        y(i,j) = (H1+H*gsi(i,j))*eta(i,j);
        gsi_x(i,j) = 1;
        gsi_y(i,j) = 0;
        eta_x(i,j) = -H*eta(i,j)/(H1+H*gsi(i,j));
        eta_y(i,j) = 1/(H1+H*gsi(i,j))
    end
end
```

The grid system is generated as follows. The geometry in the physical space is defined. For this problem, it is accomplished by specifying values of *L, H_1* , and *H_2*. Then, the desired number of grid points defined by *IM* (the maximum number of grid points in ξ) and *JM* (the maximum number of grid points in η) is specified. The equal grid spacing in the computational domain is produced as follows:

$$\Delta\xi = \frac{L}{IM - 1} \tag{9.25}$$

$$\Delta\eta = \frac{1.0}{JM - 1} \tag{9.26}$$

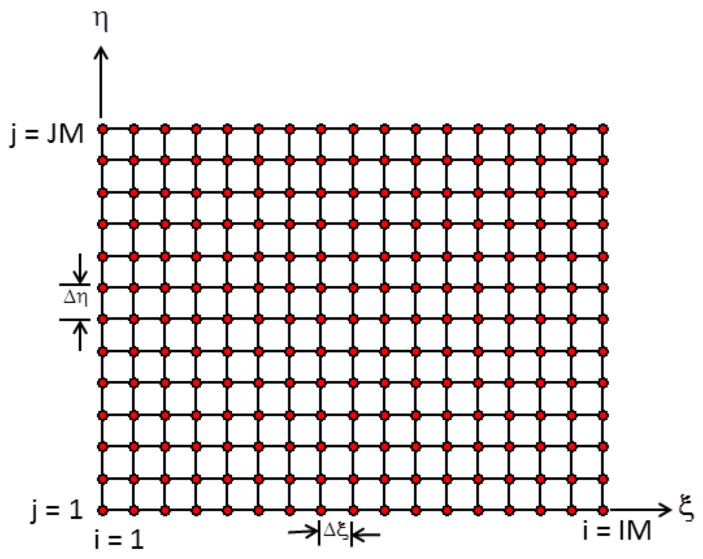


Figure 9.5 The rectangular computational domain with uniform grid spacing

Note that in Equation (9-20), η has been normalized, i.e., its value varies from zero to one. With the equal step-sizes provided by Equations (9-25) and (9-26), the uniform computational domain is constructed, which is shown in Figure 9-5 . Thus, the values of ξ and η are known at each grid point within this domain. Now Equations (9-23) and (9-24) are used to identify the corresponding grid points in the physical space. For illustrative purposes, the grid system generated for a physical domain defined by L = 4, H_1 = 2, and H_2 = 4 is shown in Figure 9-4.

The metrics and the Jacobian of the transformation must be evaluated before any transformed PDEs can be solved. In many instances, when an algebraic model is used the metrics may be calculated analytically. This aspect is obviously an advantage of the algebraic methods, since numerical computation of the metrics will require additional computation time and may introduce some errors into the system of equations of motion that are to be solved. It is strongly recommended that the metric distributions be investigated before solving the governing PDEs of fluid motion.

$$\xi_x = 1 \tag{9.27}$$

$$\xi_y = 0 \tag{9.28}$$

Similarly, the following is obtained from Equation (9.22)

$$\eta_x = -\frac{(H_2 - H_1)y/L}{[H_1 + (H_2 - H_1)x/L]^2} \qquad \text{or}$$

$$\eta_x = -\frac{H_2 - H_1}{L}\frac{\eta}{[H_1 + (H_2 - H_1)\xi/L]} \qquad (9.29)$$

and

$$\eta_y = \frac{1}{[H_1 + (H_2 - H_1)x/L]} \qquad \text{or}$$

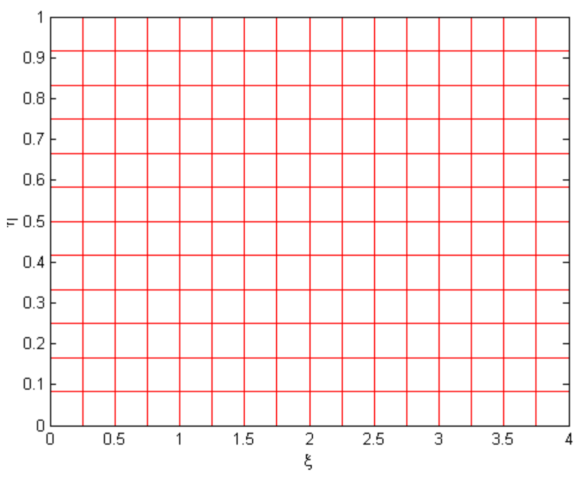
$$\eta_y = \frac{1}{[H_1 + (H_2 - H_1)\xi/L]} \qquad (9.30)$$

and the Jacobian of transformation is $J = \dfrac{1}{x_\xi y_\eta - y_\xi x_\eta}$

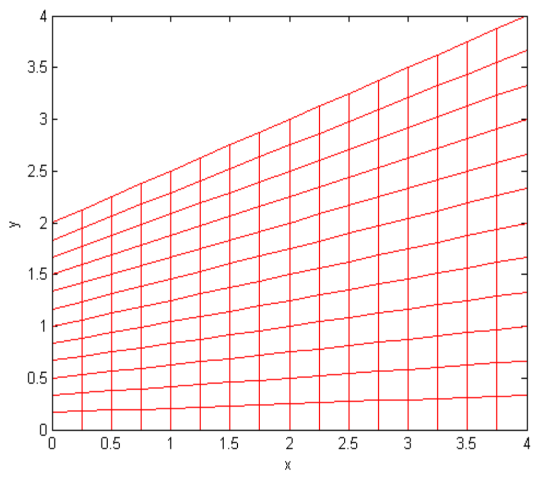
The matlab code along with distribution of metrics of the given transformation is given below:

(grid_transformation.m)

```
clc
clear all
close all

L = 4;
H1 = 2;
H2 = 4;
IM = 17;
JM = 13;
dgsi = L/(IM-1);
deta = 1/(JM-1);
H = (H2-H1)/L;

for i = 1:IM
    for j = 1:JM
%         computing the coordinates of computational domain
        gsi(i,j) = (i-1)*dgsi;
        eta(i,j) = (j-1)*deta;

%         computing the coordinates of physical domain acoording to the
%         relation given by (9.23) and (9.24)
        x(i,j) = gsi(i,j);
        y(i,j) = (H1+H*gsi(i,j))*eta(i,j);

%        computing metrics using equations(9.27),(9.28),(9.29) and (9.30)
        gsi_x(i,j) = 1;
        gsi_y(i,j) = 0;
        eta_x(i,j) = -H*eta(i,j)/(H1+H*gsi(i,j));
        eta_y(i,j) = 1/(H1+H*gsi(i,j));

    end
end

% plotting coordinates of computational domain
figure(1)
 plot(gsi,eta,'-r',gsi',eta','-r')
 xlabel('\xi')
 ylabel('\eta')


% plotting coordinates of physical domain
figure(2)
plot(x,y,'-r',x',y','-r')
xlabel('x')
ylabel('y')

% plotting distribution of metrics
figure(3)
plot3(gsi,eta,gsi_x,'-r',gsi',eta',gsi_x','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\xi_{x}')
view(36,54)

figure(4)
plot3(gsi,eta,gsi_y,'-r',gsi',eta',gsi_y','-r')
axis([0 4 0 1 0 0.5])
xlabel('\xi')
ylabel('\eta')
zlabel('\xi_{y}')
view(36,54)

figure(5)
plot3(gsi,eta,eta_x,'-r',gsi',eta',eta_x','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\eta_{x}')
set(gca,'ZDir','Reverse')
view(36,54)

figure(6)
plot3(gsi,eta,eta_y,'-r',gsi',eta',eta_y','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\eta_{y}')
view(36,54)
```

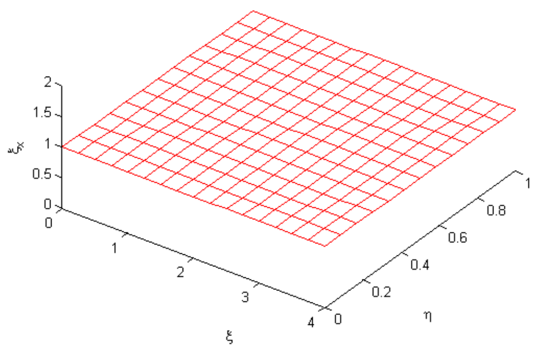


(a)

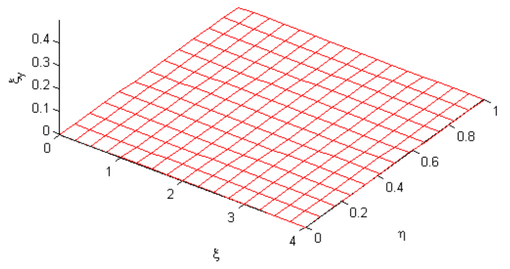


(b)

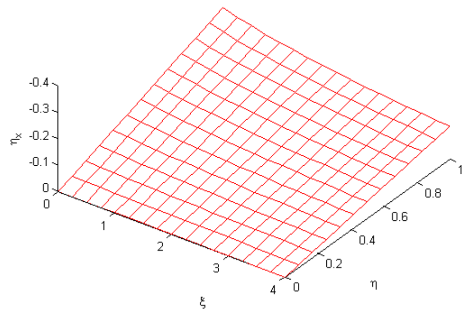
Figure 9.6 Computational and Physical Domain



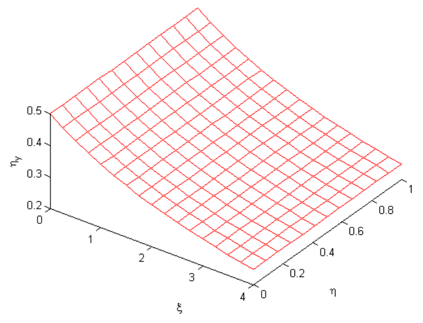
(a)



(b)



(c)



(d)

Figure 9.7a,b,c,d. Distribution of metrics for the domain shown in Figure 9. 6b

The simple algebraic model that was studied above does not include an option for clustering. The next model introduces some algebraic expressions which employ clustering techniques. For flow problems where large gradients are concentrated in a specific region, additional resolution of the flow properties is essential. As an example, flow in the vicinity of a solid surface in a viscous fluid possesses large flow gradients. Accurate computation of flow gradients in this region will require many grid points within the domain. Rather than using a nearly uniform grid distribution in the physical domain, grid points may be clustered in the regions of high flow gradients, which reduces the total number of grid points and thus increases efficiency. Some examples of such algebraic expressions with clustering options are provided below. As a first example, consider the transformation given by

$$\xi = x \tag{9.31}$$

$$\eta = 1 - \frac{\ln\left\{[\beta + 1 - (y/H)]/[\beta - 1 + (y/H)]\right\}}{\ln[(\beta+1)/(\beta-1)]} \tag{9.32}$$

In this equation, β is the clustering parameter within the range of 1 to ∞. As the value of β approaches 1, more grid points are clustered near the surface, where y = 0. From Equations (9.31) and (9.32), the inverse of the transformation can be written as

$$x = \xi \tag{9.33}$$

$$y = H \frac{(\beta+1) - (\beta-1)\left\{[(\beta+1)/(\beta-1)]^{1-\eta}\right\}}{[(\beta+1)/(\beta-1)]^{1-\eta} + 1} \tag{9.34}$$

The metrics of transformation may be determined analytically from the algebraic relations (9.31) and (9.32) and are given below:

$$\xi_x = 1$$

(9.35)

$$\eta_x = 0 \tag{9.36}$$

$$\xi_y = 0 \tag{9.37}$$

$$\eta_y = \frac{2\beta}{H\left\{\beta^2 - [1-(y/H)]^2\right\}\left\{\ln[(\beta+1)/(\beta-1)]\right\}}$$

(9.38)

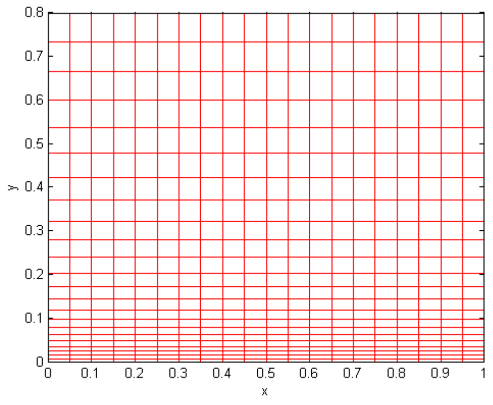
Pseudo code :

```
beta1 = (beta+1)/(beta-1);
for i = 1:IM
    for j = 1:JM
        xi(i,j) = (i-1)*dxi;
        eta(i,j) = (j-1)*deta;
        beta2 = beta1^(1-eta(i,j));
        x(i,j) = xi(i,j);
        y(i,j) = H*((beta+1)-((beta-1)*beta2))/(beta2+1);
        y_term  = (1-(y(i,j)/H))^2;
        eta_y(i,j) = 2*beta/(H*(beta*beta-y_term)*log(beta1));

    end
end
```

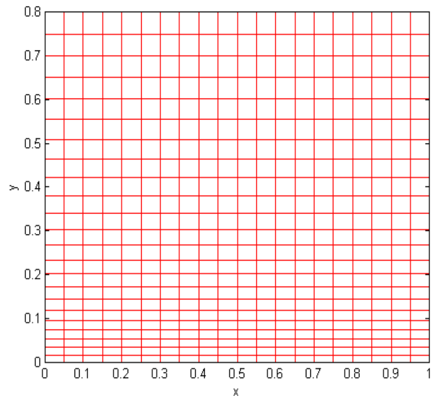
The matlab code to generate computational domain , physical domain and its metrices is given below:

(cluster_technique.m)

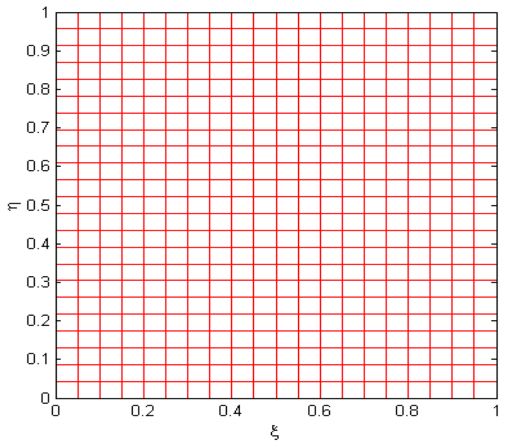
```
clc
clear all
close all

l_xi = 1.0;
l_eta = 1.0;
H = 0.8;
IM = 21;
JM = 24;
dxi = l_xi/(IM-1);
deta = l_eta/(JM-1);
beta = 1.2;                              % clustering parameter
beta1 = (beta+1)/(beta-1);
for i = 1:IM
    for j = 1:JM
%         compute coordinates of computational domain
        xi(i,j) = (i-1)*dxi;
        eta(i,j) = (j-1)*deta;
        beta2 = beta1^(1-eta(i,j));
%         computing the coordinates of physical domain acoording to the
%         relation given by (9.33) and (9.34)
        x(i,j) = xi(i,j);
        y(i,j) = H*((beta+1)-((beta-1)*beta2))/(beta2+1);
        y_term  = (1-(y(i,j)/H))^2;

 %        computing metrics using equations(9.38)
        eta_y(i,j) = 2*beta/(H*(beta*beta-y_term)*log(beta1));
    end
end

% Plotting computational domain
figure(1)
  plot(xi,eta,'-r',xi',eta','-r')
  axis image
  xlabel('\xi')
ylabel('\eta')

% Plotting physical domain
  figure(2)
  plot(x,y,'-r',x',y','-r')
  axis image
  xlabel('x')
ylabel('y')

% Plotting distribution of metrics
  figure(3)
  plot3(xi,eta,eta_y,'-r',xi',eta',eta_y','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\eta_{y}')
view(111,60)
```



(a) $\beta = 1.05$



(b) $\beta = 1.2$



(c) Computational domain

Figure 9.8a,b,c. Physical and computational domains generated by transformation functions (9.31) and (9.32)

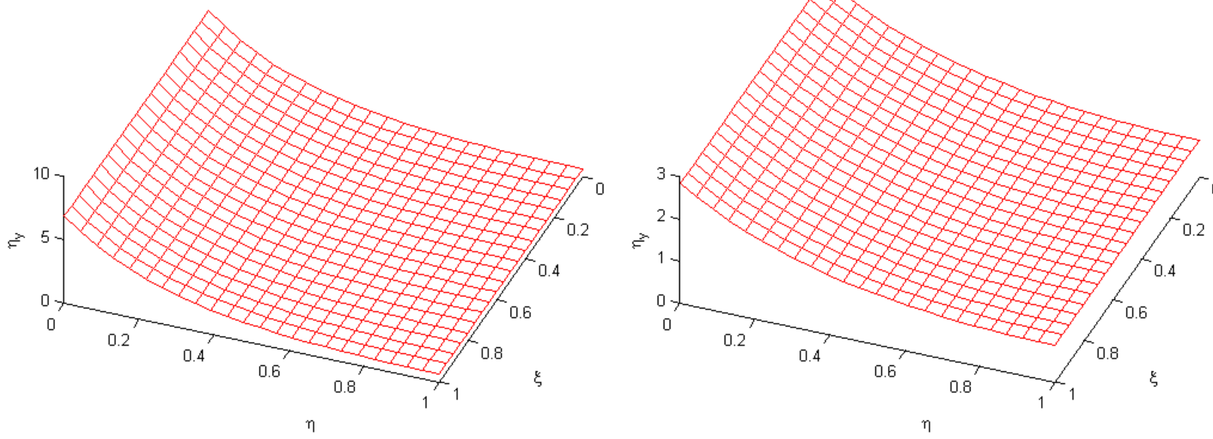


Figure 9.9a,b. Metric distribution for the domain of Figures 9.8a and 9.8b

The physical and computational domains, using the transformations (9.31) and (9.32), are shown in Figures 9.8a, 9.8b, and 9.7c. The grid system is generated for a 21 x 24 grid, and clustering parameter values of 1.05 and 1.2 are used, respectively. The distributions of metric η_y are shown in Figures 9.8a and 9.8b. This type of grid is suitable for boundary-layer type computations, where grid clustering near the surface is required.

For a physical domain enclosed by lower and upper solid surfaces, clustering at both surfaces must be considered. A flow field within a 2-D duct is such an example. The following algebraic equations may be employed for this purpose:

$$\xi = x \tag{9.39}$$

$$\eta = \alpha + (1-\alpha)\frac{\ln\left\{\{[\beta + [(2\alpha+1)y/H] - 2\alpha\}/\{\beta - [(2\alpha+1)y/H] + 2\alpha\}\right\}}{\ln[(\beta+1)/(\beta-1)]} \tag{9.40}$$

where β is the clustering parameter, and α defines where the clustering takes place. When α = 0, the clustering is at *y = H*; whereas, when α = 1/2, clustering is distributed equally at *y = 0* and *y= H*. The inverse transformation is given by

$$x = \xi \tag{9.41}$$

$$y = H\frac{(2\alpha+\beta)[(\beta+1)/(\beta-1)]^{(\eta-\alpha)/(1-\alpha)}+2\alpha-\beta}{(2\alpha+1)\{1+[(\beta+1)/(\beta-1)]^{(\eta-\alpha)/(1-\alpha)}\}} \tag{9.42}$$

Analytical expressions for the metrics are determined from Equations (9.39) and (9.40) as

$$\xi_x = 1 \tag{9.43}$$

$$\xi_y = 0 \tag{9.44}$$

$$\eta_x = 0 \tag{9.45}$$

$$\eta_y = \frac{2\beta(2\alpha+1)(1-\alpha)}{H\{\beta^2-[(2\alpha+1)y/H-2\alpha]^2\}\ln[(\beta+1)/(\beta-1)]} \tag{9.46}$$

Pseudo code :

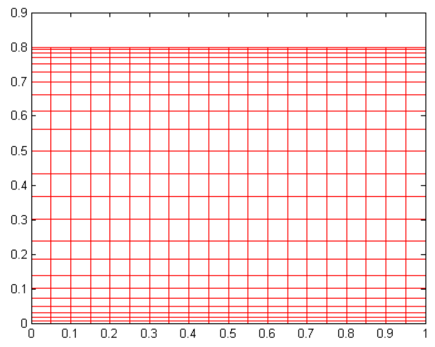
```
for i = 1:IM
    for j = 1:JM
   %compute coordinates of computational domain
        xi(i,j) = (i-1)*dxi;
        eta(i,j) = (j-1)*deta;
  %computing the coordinates of physical domain acoording to the
  %relation given by (9.41) and (9.42)
        expo = (eta(i,j)-alpha)/(1-alpha);
        beta2 = beta1^expo;
        x(i,j) = xi(i,j);
        y(i,j) = H*(bet_alph*beta2+2*alpha-beta)/((2*alpha+1)*(1+beta2));
   % computing metrics using equations(9.46)
        y_term  = ((2*alpha+1)*(y(i,j)/H)-2*alpha)^2;
        eta_y(i,j) = (2*beta*(2*alpha+1)*(1-alpha))/(H*(beta*beta-
y_term)*log(beta1));

    end
end
```

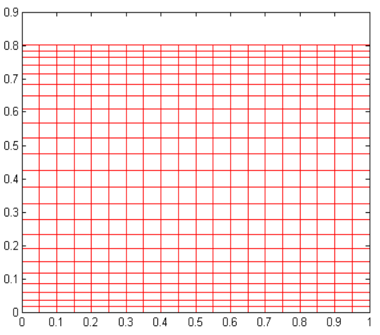
The matlab code to generate computational domain , physical domain and its metrices is given below:

(cluster_top_bottom.m)

```
clc
clear all
```

```
close all

l_xi = 1.0;
l_eta = 1.0;
H = 0.8;
IM = 21;
JM = 24;
dxi = l_xi/(IM-1);
deta = l_eta/(JM-1);
beta = 1.05;                                    %clustering parameter
alpha = 0.5;
beta1 = (beta+1)/(beta-1);
bet_alph = 2*alpha+beta;
for i = 1:IM
    for j = 1:JM
   %compute coordinates of computational domain
        xi(i,j) = (i-1)*dxi;
        eta(i,j) = (j-1)*deta;
  %computing the coordinates of physical domain acoording to the
  %relation given by (9.41) and (9.42)
        expo = (eta(i,j)-alpha)/(1-alpha);
        beta2 = beta1^expo;
        x(i,j) = xi(i,j);
        y(i,j) = H*(bet_alph*beta2+2*alpha-beta)/((2*alpha+1)*(1+beta2));
   % computing metrics using equations(9.46)
        y_term  = ((2*alpha+1)*(y(i,j)/H)-2*alpha)^2;
        eta_y(i,j) = (2*beta*(2*alpha+1)*(1-alpha))/(H*(beta*beta-
y_term)*log(beta1));

    end
end

% Plot computational domain
figure(1)
  plot(xi,eta,'-r',xi',eta','-r')
  axis image


% Plot physical domain
  figure(2)
  plot(x,y,'-r',x',y','-r')
axis([0 1 0 0.9])

% Plot distribution of metrics
  figure(3)
  plot3(xi,eta,eta_y,'-r',xi',eta',eta_y','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\eta_{y}')
view(-66,54)
```


(a) $\beta = 1.05$


(b) $\beta = 1.2$

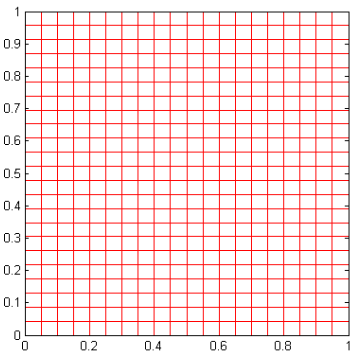
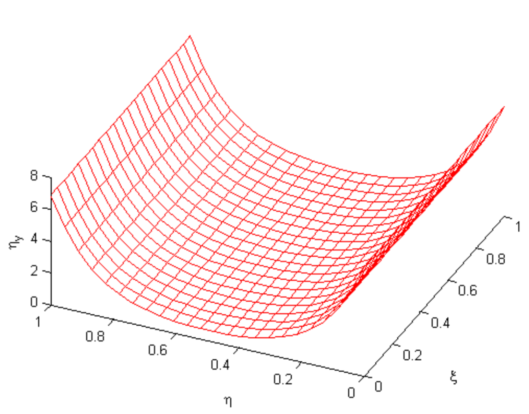

(c) Computational domain

Figure 9.10a,b,c. Physical and computational domains generated by transformation functions (9.39) and (9.40)



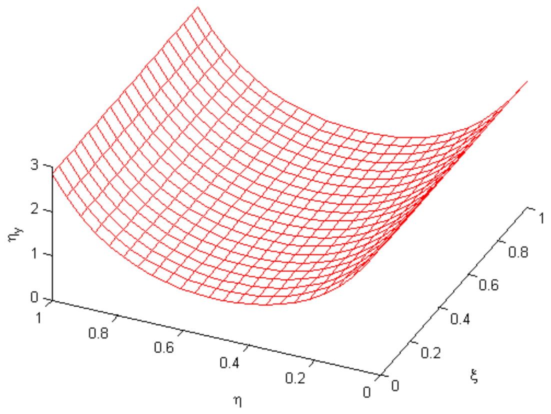


Figure 9.11a,b. Metric distribution for the domain of Figures 9.10a and 9.10b

Grid systems generated for a domain with 21 x 24 grid points are shown in Figures 9.10a and 9.10b for α = 1/2 and clustering parameters of 1.05 and 1.2, respectively. The corresponding η_y metric distributions are shown in Figures 9.11a and 9.11b.

For problems where clustering in the interior of the domain is required, the following relations can be utilized

$$\xi = x \tag{9.47}$$

$$\eta = A + \frac{1}{\beta}\sinh^{-1}\left[\left(\frac{y}{D} - 1\right)\sinh(\beta A)\right] \tag{9.48}$$

where

$$A = \frac{1}{2\beta}\ln\left[\frac{1+(e^{\beta}-1)(D/H)}{1+(e^{-\beta}-1)(D/H)}\right]$$

In Equation (9.48), β is the clustering parameter in the range of $0 < \beta < \infty$, and D is the y coordinate where clustering is desired. The inverse transformation is given by:

$$x = \xi \tag{9.49}$$

$$y = D\left\{1 + \frac{\sinh[\beta(\eta - A)]}{\sinh(\beta A)}\right\} \tag{9.50}$$

Note that for $\beta = 0$, no clustering is enforced, while a denser clustering of grid points near D is produced for larger values of β. Analytical expressions for the metrics are determined from Equations (9.47) and (9.48) and are given by:

$$\xi_x = 1 \tag{9.51}$$

$$\xi_y = 0 \tag{9.52}$$

$$\eta_x = 0 \tag{9.53}$$

$$\eta_y = \frac{\sinh(\beta A)}{\beta D\{1+[(y/D)-1]^2\sinh^2(\beta A)\}^{0.5}} \tag{9.54}$$

Pseudocode:

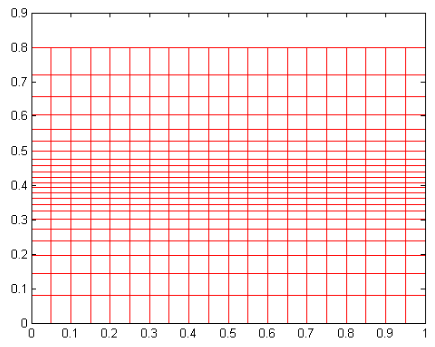
```
for i = 1:IM
    for j = 1:JM
  %  compute coordinates of computational domain
        xi(i,j) = (i-1)*dxi;
        eta(i,j) = (j-1)*deta;
  %compute the coordinates of physical domain acoording to the
  %relation given by (9.49) and (9.50)
        x(i,j) = xi(i,j);
        y(i,j) = D*(1+sinh(beta*(eta(i,j)-A))/sinh(beta*A));
   % computing metrics using equations(9.54)
        y_term  = 1+(((y(i,j)/D)-1))^2*(sinh(beta*A))^2;
        eta_y(i,j) = sinh(beta*A)/(beta*D*y_term^0.5);

    end
end
```

The matlab code to generate computational domain , physical domain and its metrices is given below:

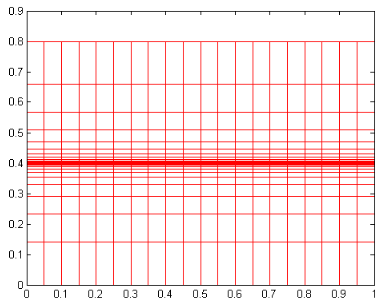
(cluster_interior_domain.m)

```
clc
clear all
close all

l_xi = 1.0;
l_eta = 1.0;
H = 0.8;
D = H/2;                                     % y-coordinate where clustering is
desired
IM = 21;
JM = 24;
dxi = l_xi/(IM-1);
deta = l_eta/(JM-1);
beta =5;                                     % clustering parameter
D_H = D/H;
a = (1+(exp(beta)-1)*D_H)/(1+(exp(-beta)-1)*D_H)
A = 0.5/beta*log(a)

for i = 1:IM
    for j = 1:JM
  %  compute coordinates of computational domain
        xi(i,j) = (i-1)*dxi;
        eta(i,j) = (j-1)*deta;
  %compute the coordinates of physical domain acoording to the
  %relation given by (9.49) and (9.50)
        x(i,j) = xi(i,j);
        y(i,j) = D*(1+sinh(beta*(eta(i,j)-A))/sinh(beta*A));
   % computing metrics using equations(9.54)
        y_term  = 1+(((y(i,j)/D)-1))^2*(sinh(beta*A))^2;
        eta_y(i,j) = sinh(beta*A)/(beta*D*y_term^0.5);

    end
end


% Plot computational domain
figure(1)
  plot(xi,eta,'-r',xi',eta','-r')
  axis image
```

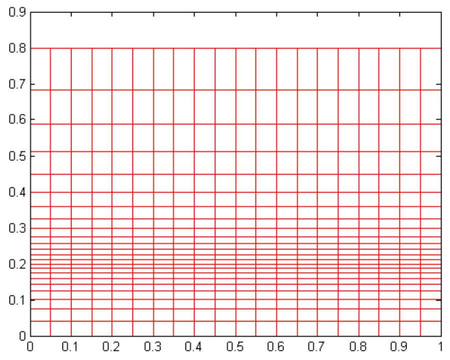
```
% Plot physical domain
  figure(2)
  plot(x,y,'-r',x',y','-r')
axis([0 1 0 0.9])

% Plot distribution of metrics
  figure(3)
  plot3(xi,eta,eta_y,'-r',xi',eta',eta_y','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\eta_{y}')
view(-70,40)
```



(a) D = H/2, $\beta = 5$

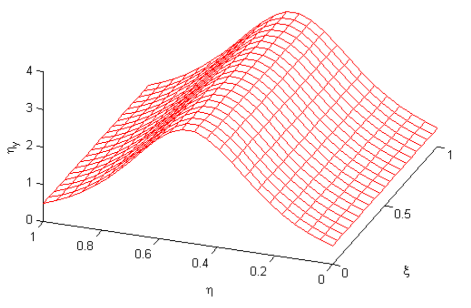


(b) D= H/2 $\beta = 10$

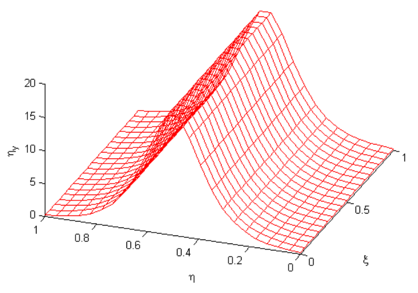


(c) D = H/4, $\beta = 5$

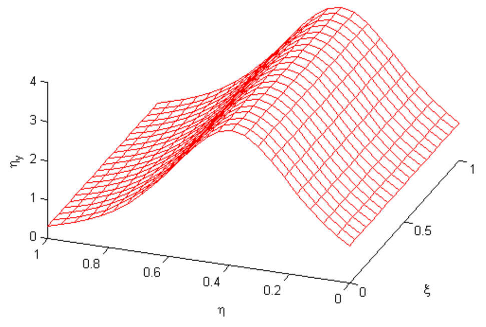
Figure 9.12a,b,c. Grid systems generated by transformation functions (9.47) and (9.48)



(a)



(b)



(c)

Figure 9.13a, b, and c. Metric distributions for the domains shown in Fig 9.12.

The algebraic expressions given by (9.49) and (9.50) are used to generate the grid systems shown in Figures 9.12a through 9.12c. The clustering is specified at D = H/2 for the domains shown in Figures 9.12a and 9.12b and at D = H/4 for the domain shown in Figure 9.12c. The values of β are 5, 10, and 5, respectively. The distributions of metric η_y are illustrated in Figures 9.13a through 9.13c.

Next, an algebraic expression for generating a grid system around an arbitrary shape is studied. For simplicity, a conical body shape with a circular cross section is taken. The grid system is determined at cross sections where the relevant PDEs are to be solved. An example would be the solution of parabolized Navier-Stokes equations over a conical configuration in a supersonic flow field. The flow field is depicted in Figure 9.14 where cross-sectional planes are normal to the body axis.

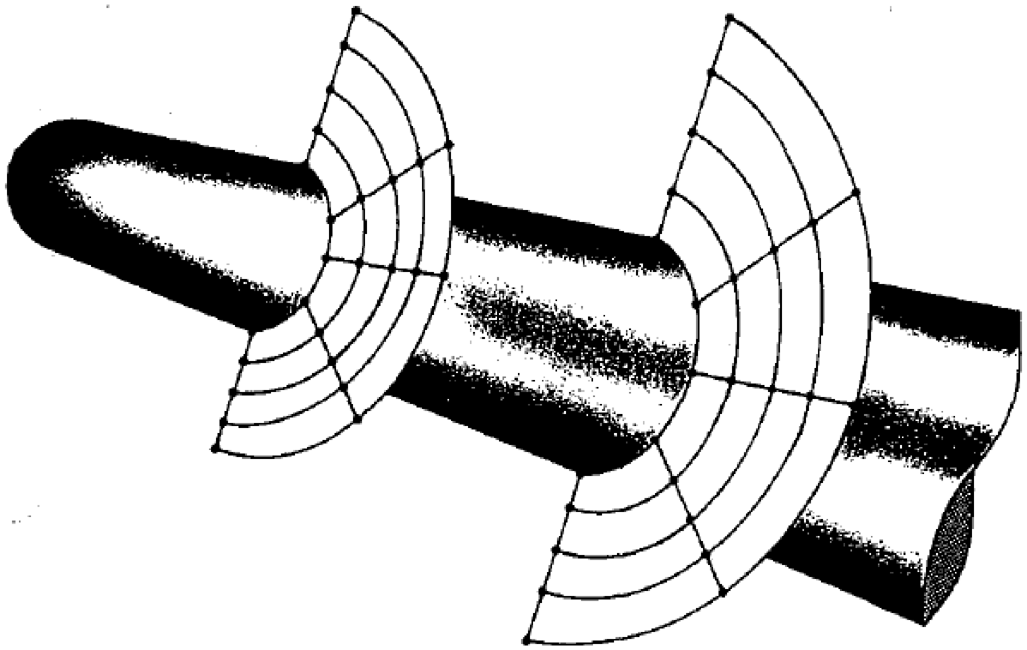


Figure 9.14 Grid system in the physical domain

An alternate choice would be the selection of a grid system normal to the body surface. For many applications where the flow is symmetrical, only half of the domain needs to be solved. The generation of a grid system at a cross section is considered. Note that since the physical domain is changing at each streamwise station, a new grid system must be generated at each station.

Here, the outer boundary is specified as the free stream. This specification is accomplished by defining two elliptical with semi-major and semi-minor axes denoted by a_1, b_1, and b_2. This specification is illustrated in Figure 9.15. The inner boundary represents the circular cone and is defined by its radius, R.

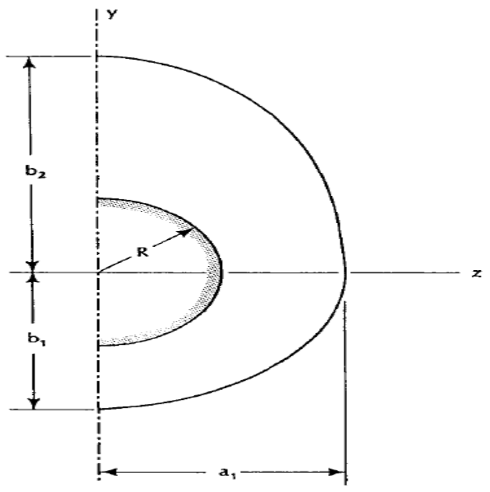


Figure 9.15 Nomenclature to describe physical domain

The number of grid points in the circumferential direction is specified by *KM*, and for this example they are distributed equally around the body. This distribution is accomplished by defining the incremental angle DELTHET as DELTHET = π/(KM- 1), and subsequently computing θ at each *k*. The nomenclature is shown in Figure 9.16. Note that *k = 1* is chosen on the windward side of the body and *k = KM* is on the leeward side.

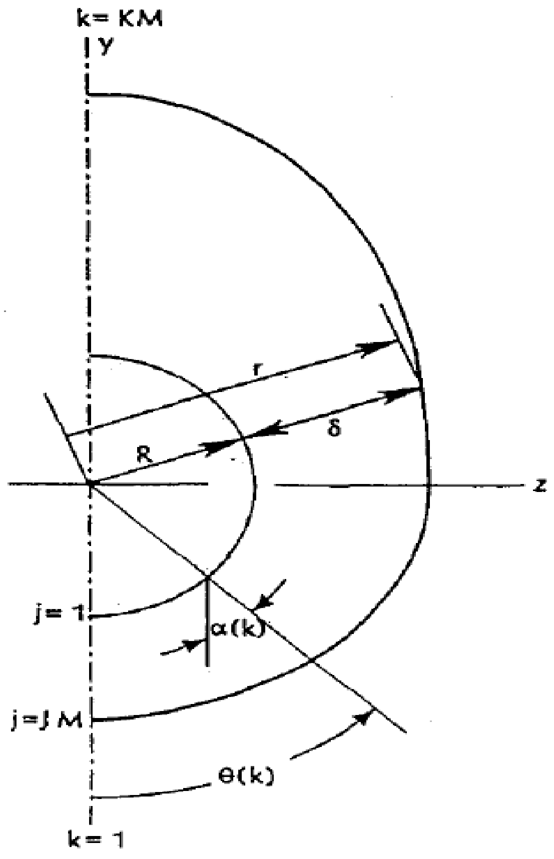


Figure 9.16 Nomenclature to define the grid system

The y and z coordinates of the grid points on the body are easily computed from

$$y(1,k) = -R\cos\theta \qquad (9.55)$$

$$z(1,k) = R\sin\theta \qquad (9.56)$$

The corresponding grid points on the outer boundary are defined by rays emanating from the origin with the appropriate angular positions.

The length of rays are obtained from the equation of an ellipse as

$$r = \frac{1}{\sqrt{\left(\frac{\sin^2\theta}{a^2}\right)+\left(\frac{\cos^2\theta}{b^2}\right)}}$$

where appropriate values of *a* and *b* are used. Subsequently, the *y* and *z* coordinates of the grid points on the outer boundary are determined by similar expressions given in Equations (9.55) and (9.56). For viscous flow computations, clustering in the vicinity of the body is desirable. For this purpose the following expression is used:

$$c(k,j) = \delta\left\{1-\beta\left[\left(\frac{\beta+1}{\beta-1}\right)^{\eta}-1\right]\Big/\left[\left(\frac{\beta+1}{\beta-1}\right)^{\eta}+1\right]\right\} \quad (9.57)$$

In (9. 57) , δ is the radial distance between the body and the outer boundary, i.e.,

$$\delta(k) = r(k) - R(k) \quad (9.\ 58)$$

and β is the clustering parameter. The clustering function given by Equation (9.57) places η = 0 at the outer boundary and η= 1.0 at the surface. The *y* and *z* coordinates within the physical domain are evaluated from the following equations

$$y(k,j) = y(k,1) - c(k,j)\cos\alpha(k) \quad (9.59)$$

and

$$z(k,j) = z(k,1) + c(k,j)\sin\alpha(k) \quad (9.60)$$

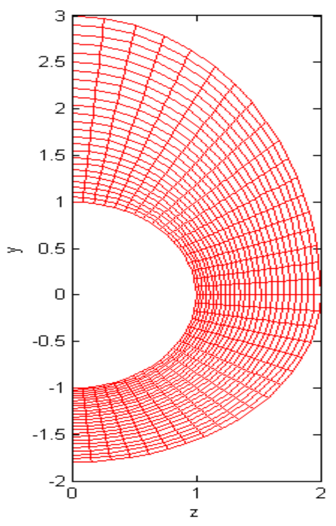
where α is defined as the angle between the normal to the body in the cross-sectional plane and the vertical direction.

The matlab code to generate computational domain , physical domain and its metrics is given below:

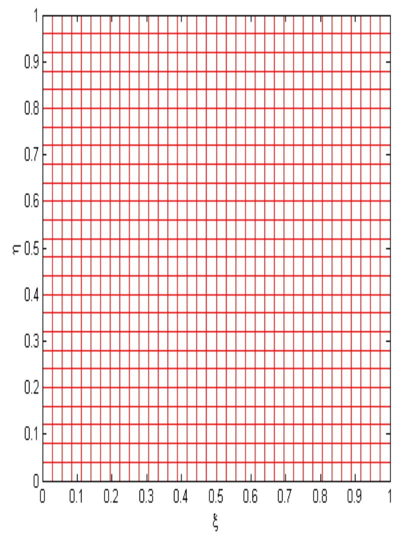
(cluster_interior_domain.m)

```
clc
clear all
close all

R = 1.0;
a1 = 2.0;
b1 = 1.8;
b2 = 3.0;
beta = 1.;
beta1 = (beta+1)/(beta-1);

l_xi = 1.0;
l_eta = 1.0;
KM = 37;
```

```
JM = 26;
dxi = l_xi/(KM-1);
deta = l_eta/(JM-1);
delthet = pi/(KM-1);

for k = 1:KM
    for j = 1:JM
%           compute coordinates of computational domain
        xi(k,j) = (k-1)*dxi;
        eta(k,j) = (JM-j)*deta;

 %          compute coordinates of physical domain using equations
(9.55),(9.56),(9.57) ,(9.58),(9.59) and (9.60)
        theta = (k-1)*delthet;
        beta2 = beta1^(eta(k,j));
        if j == 1
            y(k,1) = -R*cos(theta);
            z(k,1) = R*sin(theta);
        end
        if k<=KM/2
            r = 1/sqrt((sin(theta)^2/a1^2)+(cos(theta)^2/b1^2));
        else
            r = 1/sqrt((sin(theta)^2/a1^2)+(cos(theta)^2/b2^2));
        end
        delta = r-R;
        c(k,j) = delta*(1-((beta*(beta2-1))/(beta2+1)));
        y(k,j) = y(k,1)-c(k,j)*cos(theta);
        z(k,j) = z(k,1)+c(k,j)*sin(theta);
    end
end

% compute metric distribution numerically
for k = 1:KM
    for j = 1:JM
        if k == 1
        z_xi = (-3*z(k,j)+4*z(k+1,j)-z(k+2,j))/(2*dxi);
        y_xi = (-3*y(k,j)+4*y(k+1,j)-y(k+2,j))/(2*dxi);

        elseif k == KM
        z_xi = (3*z(k,j)-4*z(k-1,j)+z(k-2,j))/(2*dxi);
        y_xi = (3*y(k,j)-4*y(k-1,j)+z(k-2,j))/(2*dxi);

        else
        z_xi = (z(k+1,j)-z(k-1,j))/(2*dxi);
        y_xi = (y(k+1,j)-y(k-1,j))/(2*dxi);
        end



        if j == 1
        z_eta = (-3*z(k,j)+4*z(k,j+1)-z(k,j+2))/(2*deta);
        y_eta = (-3*y(k,j)+4*y(k,j+1)-y(k,j+2))/(2*deta);

        elseif j == JM
        z_eta = (3*z(k,j)-4*z(k,j-1)+z(k,j-2))/(2*deta);
        y_eta = (3*y(k,j)-4*y(k,j-1)+y(k,j-2))/(2*deta);
        else
        z_eta = (z(k,j+1)-z(k,j-1))/(2*deta);
        y_eta = (y(k,j+1)-y(k,j-1))/(2*deta);
        end
        inv = (z_xi*y_eta)-(y_xi*z_eta);
        J = 1/inv;
        xi_z(k,j) = J*y_eta;
        xi_y(k,j) = -J*z_eta;
        eta_z(k,j) = -J*y_xi;
        eta_y(k,j) = J*z_xi;

    end
end

% plot physical domain
figure(1)
plot(z,y,'-r',z',y','-r')
xlabel('z')
ylabel('y')

% plot computational domain
figure(2)
plot(xi,eta,'-r',xi',eta','-r')
xlabel('\xi')
ylabel('\eta')

% plot distribution of metrics
figure(3)
plot3(xi,eta,xi_z,'-r',xi',eta',xi_z','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\xi_{z}')

figure(4)
plot3(xi,eta,xi_y,'-r',xi',eta',xi_y','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\xi_{y}')

figure(5)
plot3(xi,eta,eta_z,'-r',xi',eta',eta_z','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\eta_{z}')

figure(6)
plot3(xi,eta,eta_y,'-r',xi',eta',eta_y','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\eta_{y}')
```

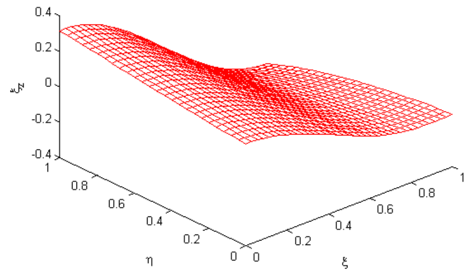


(a)

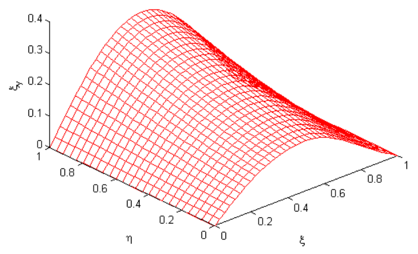


(b)

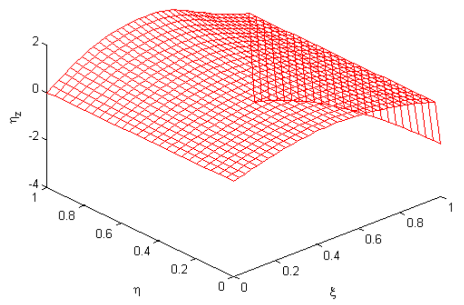
Figure 9.17a, b. Physical and Computational domain



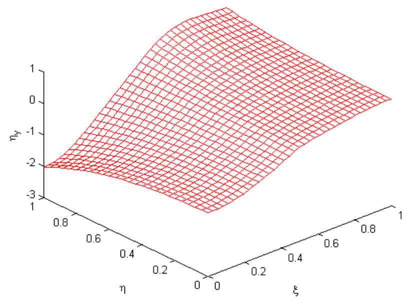
(a)



(b)



(c)



(d)

Figure 9.18a, b, c,d. Metric distributions for the domains shown in Fig 9.17.

A grid system generated for a 37 X 26 grid is shown in Figure 9.17 for β = 1.4. The configuration is defined by R = 1.0, a_1 = 2.0, b_1 = 1.8, and b_2 = 3.0. For this problem the metrics are evaluated numerically by using a second-order central difference approximation in the interior of the domain and by second-order forward and backward difference approximations at the boundaries. The metric distributions are illustrated in Figures 9.18a through 9.18d, where the smoothness of the metrics is clearly evident. Note that if the metric distribution had discontinuities, further investigation of the suggested grid system and the solution procedure used to obtain the grid and the metrics would be required. The algebraic expressions used to generate the grid systems just presented are a few among many appearing in various literature. However, the procedures used to generate grids by algebraic methods are fundamentally similar. For many applications, algebraic models provide a reasonable grid system with continuous and smooth metric distributions. However, if grid smoothness, skewness, and orthogonality are of concern, grid systems generated by solving PDEs must be used.

The advantages of the algebraic grid generation methods are:

(1) Computationally, they are very fast;

(2) Metrics may be evaluated analytically, thus avoiding numerical errors;

(3) The ability to cluster grid points in different regions can be easily implemented.

The disadvantages are:

(1) Discontinuities at a boundary may propagate into the interior region which could lead to errors due to sudden changes in the metrics;

(2) Control of grid smoothness and skewness is a difficult task.

Some of the disadvantages of the algebraic grid generators are overcome by the use of PDE grid generators which is, of course, accomplished with increased computational time. This procedure will be described in the next section.

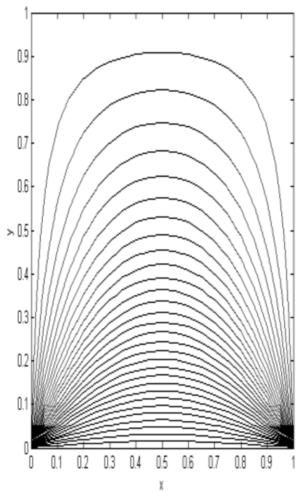
Partial Differential Equation Techniques

A grid generation scheme in which PDEs are used to create the grid system is another method. In these methods, a system of PDEs is solved for the location of the grid points in the physical space, whereas the computational domain is a rectangular shape with uniform grid spacing. These methods may be categorized as an elliptic, parabolic, or hyperbolic system of PDEs. The elliptic grid generator is the most extensively developed method. It is commonly used for 2-D problems and the procedure has been extended to 3-D problems. Parabolic and hyperbolic grid generators are not as well developed but have some very interesting features. The presentation of various schemes are limited to 2-D problems in this section.

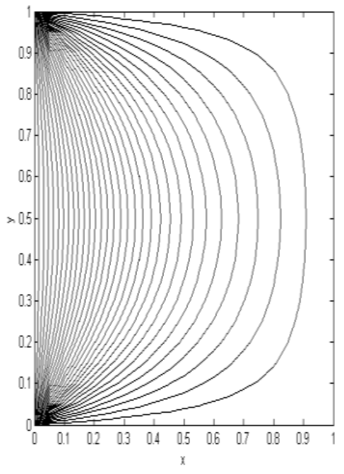
Elliptic Grid Generators

For domains where all the physical boundaries are specified, elliptic grid generators work very well. A system of elliptic equations in the form of Laplace's equation or Poisson's equation is introduced, which is solved for the coordinates of the grid points in the physical domain. Any iterative scheme such as Gauss-Seidel, point successive over relaxation (PSOR), etc., can be used to solve the elliptic PDEs. Before proceeding with the mathematical formulation, let us consider the fundamental concept behind this procedure. Recall that the heat conduction equation for a steady, 2-D problem is reduced to an elliptic PDE. If a rectangular domain with the values of temperature on the boundaries is specified, the temperature distribution within the interior points is easily obtained by any iterative scheme. The solution provides the isothermal lines. Consider two such solutions shown in Figure 9.19

.



(a)



(b)

Figure 9.19a, b. Isothermal lines for the two rectangular domains

Superimposing the two solutions yields a solution in Figure 9.20

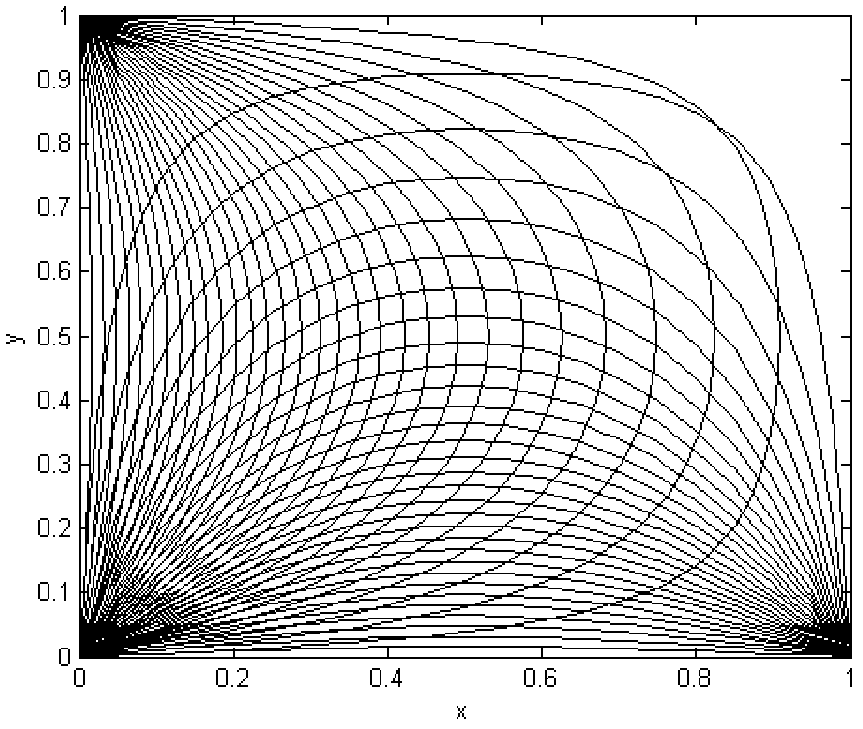


Figure 9.20 Superimposed solution for the rectangular domain

Note that the governing equation is linear; thus, addition of solution is allowed. If a heat source within the domain is specified, the isothermal lines will be altered and can be oriented to a particular region. The fundamental idea of using elliptic PDEs to generate a grid system is representation of grid lines through isothermal lines. The dependent variables are the *x* and *y* coordinates of the grid points in the physical space. Thus, for a closed domain, the distribution of grid points on the boundaries is specified and a set of elliptic PDEs is solved to locate the coordinates of the interior grid points. Consider a system of elliptic PDEs of the form

$$\xi_{xx} + \xi_{yy} = 0 \tag{9.61}$$

$$\eta_{xx} + \eta_{yy} = 0 \tag{9.62}$$

where ξ and η represent the coordinates in the computational domain. Equations (9.61) and (9.62) can be solved by any of the iterative techniques. However, computation must take place in a rectangular domain with uniform grid spacing as described earlier. To transform the elliptic PDEs, the dependent and independent variables are interchanged. The elliptic equations (9.61) and (9.62) become:

(Note the derivation of the following equation can be referred from Computational Fluid Dynamics by Hoffman (Appendix F))

$$a x_{\xi\xi} - 2b x_{\xi\eta} + c x_{\eta\eta} = 0 \tag{9.63}$$

$$a y_{\xi\xi} - 2b y_{\xi\eta} + c y_{\eta\eta} = 0 \tag{9.64}$$

where

$$a = x_n^2 + y_n^2 \tag{9.65}$$

$$b = x_\xi x_n + y_\xi y_n \tag{9.66}$$

$$c = x_\xi^2 + y_\xi^2 \tag{9.67}$$

The system of elliptic equations (9.63) and (9.64) is solved in the computational domain (ξ, η) in order to provide the grid point locations in the physical space (*x, y*). Note that the equations are nonlinear; thus, a linearization procedure must be employed. For simplicity, lagging of the coefficients will be used, i.e., the coefficients *a*, *b*, and *c* are evaluated at the previous iteration level.

Three categories of physical domains are considered here. They are: (1) a simply-connected domain, (2) a doubly-connected domain, and (3) a multiplyconnected domain. The description of each domain, specification of boundary points (conditions), examples, and analyses are given below.

Simply-Connected Domain

By definition a simply-connected region is one which is reducible and can be contracted to a point. Thus, for a simply-connected region, there are no objects within the domain. An example of a simply-connected domain is shown in Figure 9.21a. The corresponding computational domain is shown in Figure 9.21b.

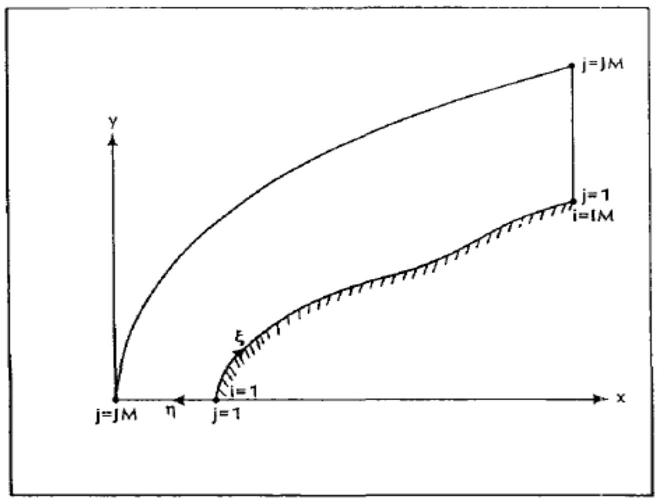


Figure 9.21a Physical domain for simply-connected region

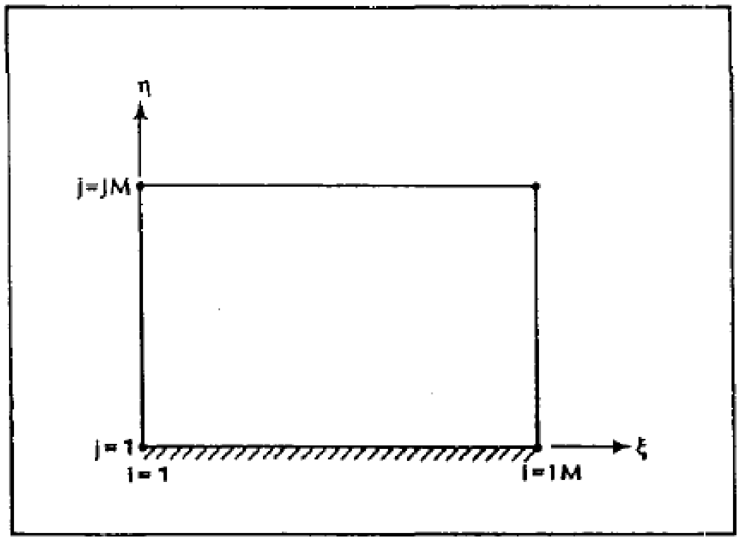


Figure 9.21b Computational domain

Another example is shown in Figure 9.22a, whereas Figure 9.22b represents the corresponding computational domain.

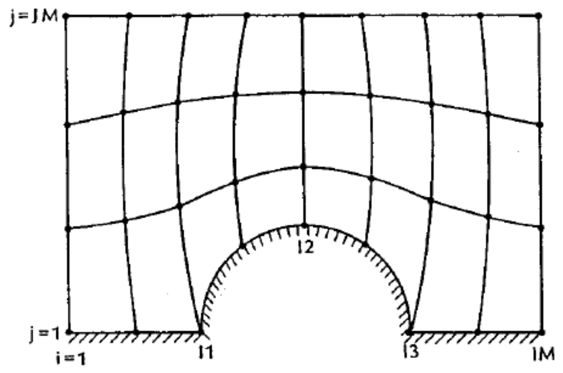


Figure 9.22a Physical domain

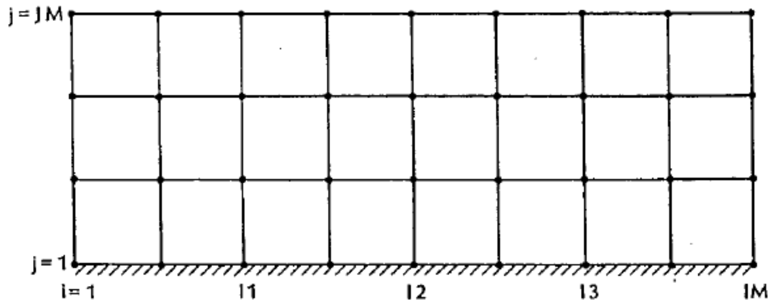


Figure 9.22b Computational domain

In order to investigate a few iterative solution schemes, the FDE is obtained by replacing the partial derivatives with a second-order central difference approximation. From Equation (9.63):

$$a\left[\frac{x_{i+1,j}-2x_{i,j}+x_{i-1,j}}{(\Delta\xi)^2}\right]-2b\left[\frac{x_{i+1,j+1}-x_{i+1,j-1}+x_{i-1,j-1}-x_{i-1,j+1}}{4\Delta\xi\Delta\eta}\right]+c\left[\frac{x_{i,j+1}-2x_{i,j}+x_{i,j-1}}{(\Delta\eta)^2}\right]=0$$

If the Gauss-Seidel iterative method is used, the equation is rearranged as:

$$2\left[\frac{a}{(\Delta\xi)^2}+\frac{c}{(\Delta\eta)^2}\right]x_{i,j}=\frac{a}{(\Delta\xi)^2}\left[x_{i+1,j}+x_{i-1,j}\right]+\frac{c}{(\Delta\eta)^2}\left[x_{i,j+1}+x_{i,j-1}\right]-\frac{b}{2\Delta\xi\Delta\eta}\left[x_{i+1,j+1}-x_{i+1,j-1}+x_{i-1,j-1}-x_{i-1,j+1}\right]$$

From which

$$x_{i,j}=\left\{\frac{a}{(\Delta\xi)^2}\left[x_{i+1,j}+x_{i-1,j}\right]+\frac{c}{(\Delta\eta)^2}\left[x_{i,j+1}+x_{i,j-1}\right]-\frac{b}{2\Delta\xi\Delta\eta}\left[x_{i+1,j+1}-x_{i+1,j-1}+x_{i-1,j-1}-x_{i-1,j+1}\right]/2\left[\frac{a}{(\Delta\xi)^2}+\frac{c}{(\Delta\eta)^2}\right]\right\} \tag{9.68}$$

Similarly, from equation (9.64)

$$y_{i,j}=\left\{\frac{a}{(\Delta\xi)^2}\left[y_{i+1,j}+y_{i-1,j}\right]+\frac{c}{(\Delta\eta)^2}\left[y_{i,j+1}+y_{i,j-1}\right]-\frac{b}{2\Delta\xi\Delta\eta}\left[y_{i+1,j+1}-y_{i+1,j-1}+y_{i-1,j-1}-y_{i-1,j+1}\right]/2\left[\frac{a}{(\Delta\xi)^2}+\frac{c}{(\Delta\eta)^2}\right]\right\} \tag{9.69}$$

Pseudo code:

```
%    Equation (9.68) & (9.69)
%     computation of coefficients a,b & c lag by one iterative level
    for i = 2:IM-1
        for j = 2:JM-1

     a = ((x(i,j+1)-x(i,j-1))/(2*deta))^2+((y(i,j+1)-y(i,j-1))/(2*deta))^2;
     b = ((x(i+1,j)-x(i-1,j))/(2*dxi))*((x(i,j+1)-x(i,j-
1))/(2*deta))+((y(i+1,j)-y(i-1,j))/(2*dxi))*((y(i,j+1)-y(i,j-1))/(2*deta));
     c = ((x(i+1,j)-x(i-1,j))/(2*dxi))^2+((y(i+1,j)-y(i-1,j))/(2*dxi))^2;
     a_1(i,j) = a/(dxi)^2;
     b_1(i,j) = b/(2*dxi*deta);
     c_1(i,j) = c/(deta)^2;
        end
    end



    for i = 2:IM-1
        for j = 2:JM-1
            ax_2 = a_1(i,j)*(x(i+1,j)+x(i-1,j));
            bx_2 = b_1(i,j)*(x(i+1,j+1)-x(i+1,j-1)+x(i-1,j-1)-x(i-1,j+1));
            cx_2 = c_1(i,j)*(x(i,j+1)+x(i,j-1));
            ay_2 = a_1(i,j)*(y(i+1,j)+y(i-1,j));
            by_2 = b_1(i,j)*(y(i+1,j+1)-y(i+1,j-1)+y(i-1,j-1)-y(i-1,j+1));
            cy_2 = c_1(i,j)*(y(i,j+1)+y(i,j-1));
            x(i,j) = (ax_2+cx_2-bx_2)/(2*(a_1(i,j)+c_1(i,j)));
            y(i,j) = (ay_2+cy_2-by_2)/(2*(a_1(i,j)+c_1(i,j)));
        end
    end
```

To start the solution, an initial distribution of *x* and *y* coordinates of the grid points within the physical domain should be provided. This distribution may be obtained by using an algebraic model. The coefficients *a, b,* and *c* appearing in Equations (9.68) and (9.69) are determined from Equations (9.65) through (9.67) using finite difference approximations. The *x* and *y* values in these expressions are provided by the initial distributions for the first iteration, and subsequently from the previous iteration, i.e., the computation of coefficients lag by one iterative level. The iterative solution continues until a specified convergence criterion is met. For this purpose the total changes in the dependent variables are evaluated as

$$\text{ERRORX} = \sum_{\substack{i=2\\j=2}}^{\substack{j=JMM1\\i=IMM1}} ABS\left[x_{i,j}^{k+1} - x_{i,j}^{k}\right]$$

$$\text{ERRORY} = \sum_{\substack{i=2\\j=2}}^{\substack{j=JMM1\\i=IMM1}} ABS\left[y_{i,j}^{k+1} - y_{i,j}^{k}\right]$$

$$\text{ERRORT=ERRORX+ERRORY}$$

where k represents the iterative level. The convergence criterion is set as ERRORT<ERRORMAX where ERRORMAX is a specified input.

Pseudo code:

```
%     Error monitoring

    errorX = 0 ;
    errorY = 0 ;


    for i = 1:IM
        for j = 1:JM
            errorX = errorX + abs(x(i,j)-prev_x(i,j));
            errorY = errorY + abs(y(i,j)-prev_y(i,j));

        end
    end
    errorT = errorX+errorY;
    count = count+1;
    iteration = iteration+1,  errorT

    if (errorT<errormax)
        break;
    end
```

Here, an example is given where the elliptic grid generator is used to create the required grid system. Consider an axisymmetric blunt body at zero degree angle of attack. The configuration is an elliptical shape defined by the semi-major axis a_1 and semi-minor axis b_1. Similarly, an elliptically shaped outer boundary is defined by specifying a_2 and b_2. The nomenclature is shown in Figure 9.23.

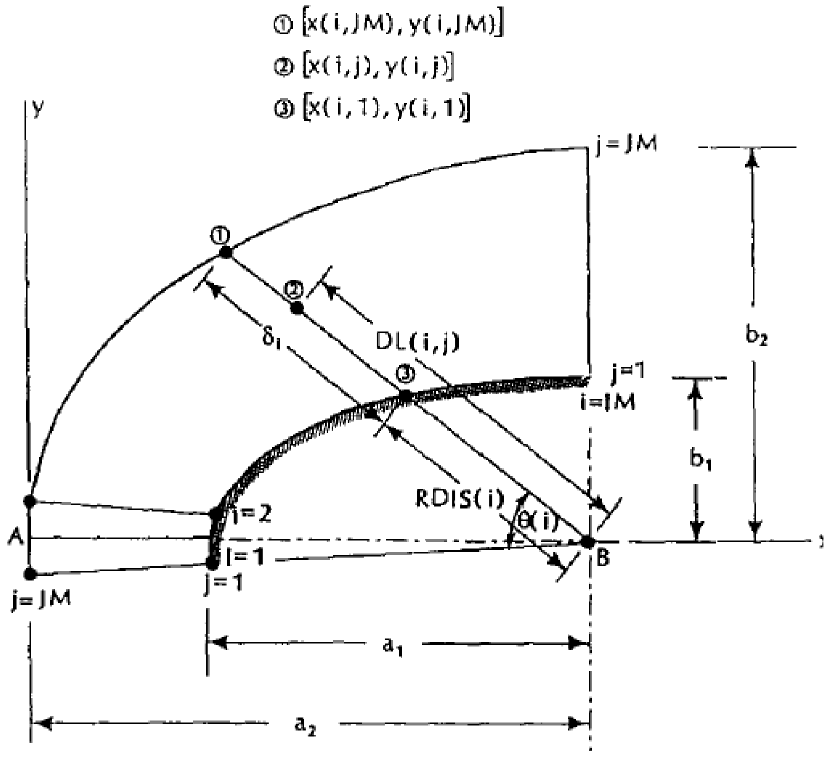


Figure 9.23 Illustration of the nomenclature for the proposed example

Since the grid system generated here is used later to solve the Euler equations, the first grid line, i.e., i = 1, is specified below the stagnation line (which is aligned along the x-axis). Symmetry of grid lines about the x-axis for grid lines i = 1 and i = 2 is enforced.

The reason for this selection of grid lines is that some difficulty in solving the Euler equations is observed if grid line i = 1 is chosen along the stagnation line. Before the elliptic equations (9.63) and (9.64) can be solved, the grid point distribution along the boundaries and an initial grid point distribution within the domain must be specified. The distribution of grid points along the boundaries may be accomplished by using various procedures. One may select equal spacing of grid points on the body or distribute them by defining an angular position θ. Clustering of points may be specified in different regions. For example, one may choose to cluster grid points in the vicinity of the stagnation point. In this problem, an equally divided angular position is used. The intersection of a ray originating from point B (Figure 9.23) at an angular position $\theta(i)$ with the body surface defines the x and y coordinate of grid point i at the surface where j = 1. Similarly, the grid point distribution on the outer boundary, where $j = JM$, is determined. In the radial direction, an algebraic expression with a clustering option is used to distribute the grid points. Since the step-size in the computational domain denoted by $\Delta\xi$ and $\Delta\eta$ can be selected arbitrarily, it is sometimes set to unity, i.e., $\Delta\xi = \Delta\eta = 1.0$.

That is the case for the algebraic expression given by

$$c(i,j) = \delta(i)\left\{1 + \beta\frac{\left[1 - \left(\frac{\beta+1}{\beta-1}\right)^{1-\gamma}\right]}{\left[1 - \left(\frac{\beta+1}{\beta-1}\right)^{1-\gamma}\right]}\right\} \tag{9.74}$$

where $\gamma = (j-1)/(JM-1)$. The radial location of grid points is determined from

$$DL(i,j) = RDIS(i,1) + c(i,j) \tag{9.75}$$

where the nomenclature is given in Figure (9.23). This procedure is also employed to distribute the grid points on the radial boundaries along $i = 2$ and $i = IM$.

The relevant data includes $a_1 = 2$, $b_1 = 1.25$, $a_2 = 3.0$, $b_2 = 3.5$, $IM = 16$, $JM = 12$, and the clustering parameter is set to 5. With the initial grid point distribution available, it is now possible to use the elliptic grid generator, i.e., Equations (9.63) and (9.64). The convergence criterion ERRORMAX is set to 0.1. The metrics ξ_x, ξ_y, η_x and η_y are computed numerically using a second-order approximate finite difference scheme. A central difference expression is used for the interior points, whereas forward and backward approximation schemes are utilized at the boundaries.

The matlab code to generate physical domain and its metrics is given below:

(simply_connected_domain_PGS.m)

```
tic
clc
clear all
close all

IM = 16;
JM = 12;
dxi = 1.0;
deta = 1.0;

for i = 1:IM
    for j = 1:JM
        xi(i,j) = (i-1)*dxi;
        eta(i,j) = (j-1)*deta;
    end
```

```
end

R = 1.0;
a1 = 2.0;
b1 = 1.25;
a2 = 3.0;
b2 = 3.5;
beta = 5;                                                    % clustering
parameter
beta1 = (beta+1)/(beta-1);
theta = linspace(0,pi/2,IM);
for i = 1:IM
    % radius of inner ellipse
    RDIS(i,1) = 1/sqrt((cos(theta(i))^2/a1^2)+(sin(theta(i))^2/b1^2));

    % radius of outer ellipse
    rdis(i,JM) = 1/sqrt((cos(theta(i))^2/a2^2)+(sin(theta(i))^2/b2^2));
    delta(i) = rdis(i,JM)-RDIS(i,1);
end

% Compute grid by the algebraic procedure
for i = 1:IM
    for j = 1:JM
        gamma = (j-1)/(JM-1);
        num = 1-(beta1)^(1-gamma);
        den = 1+(beta1)^(1-gamma);
        beta2 = num/den;
        c(i,j) = delta(i)*(1+beta*beta2);  % Equation 9.74
        DL(i,j) = RDIS(i,1)+c(i,j);        % Equation 9.75
        x(i,j) = -DL(i,j) * cos(theta(i));
        y(i,j) = DL(i,j) * sin(theta(i));
    end
end

figure(1)
plot(x,y,'-r',x',y','-r')
errormax = 0.00001;
count = 1;
iteration = 0;

% compute grid by elliptic PDE scheme
while count>0
    count = 0;
    prev_x = x;
    prev_y = y;

%    Equation (9.68) & (9.69)
%     computation of coefficients a,b & c lag by one iterative level
    for i = 2:IM-1
        for j = 2:JM-1

     a = ((x(i,j+1)-x(i,j-1))/(2*deta))^2+((y(i,j+1)-y(i,j-1))/(2*deta))^2;
     b = ((x(i+1,j)-x(i-1,j))/(2*dxi))*((x(i,j+1)-x(i,j-
1))/(2*deta))+((y(i+1,j)-y(i-1,j))/(2*dxi))*((y(i,j+1)-y(i,j-1))/(2*deta));
     c = ((x(i+1,j)-x(i-1,j))/(2*dxi))^2+((y(i+1,j)-y(i-1,j))/(2*dxi))^2;
     a_1(i,j) = a/(dxi)^2;
     b_1(i,j) = b/(2*dxi*deta);
     c_1(i,j) = c/(deta)^2;
        end
    end



    for i = 2:IM-1
        for j = 2:JM-1
            ax_2 = a_1(i,j)*(x(i+1,j)+x(i-1,j));
            bx_2 = b_1(i,j)*(x(i+1,j+1)-x(i+1,j-1)+x(i-1,j-1)-x(i-1,j+1));
            cx_2 = c_1(i,j)*(x(i,j+1)+x(i,j-1));
            ay_2 = a_1(i,j)*(y(i+1,j)+y(i-1,j));
            by_2 = b_1(i,j)*(y(i+1,j+1)-y(i+1,j-1)+y(i-1,j-1)-y(i-1,j+1));
            cy_2 = c_1(i,j)*(y(i,j+1)+y(i,j-1));
            x(i,j)  = (ax_2+cx_2-bx_2)/(2*(a_1(i,j)+c_1(i,j)));
            y(i,j)  = (ay_2+cy_2-by_2)/(2*(a_1(i,j)+c_1(i,j)));
        end
    end

%     Error monitoring
    errorX = 0 ;
    errorY = 0 ;


    for i = 1:IM
        for j = 1:JM
            errorX = errorX + abs(x(i,j)-prev_x(i,j));
            errorY = errorY + abs(y(i,j)-prev_y(i,j));

        end
    end
    errorT = errorX+errorY;
    count = count+1;
    iteration = iteration+1,  errorT

    if (errorT<errormax)
        break;
    end
end

for i = 1:IM
    for j = 1:JM
        if i == 1
        x_xi = (-3*x(i,j)+4*x(i+1,j)-x(i+2,j))/(2*dxi);
        y_xi = (-3*y(i,j)+4*y(i+1,j)-y(i+2,j))/(2*dxi);

        elseif i == IM
        x_xi = (3*x(i,j)-4*x(i-1,j)+x(i-2,j))/(2*dxi);
        y_xi = (3*y(i,j)-4*y(i-1,j)+x(i-2,j))/(2*dxi);

        else
        x_xi = (x(i+1,j)-x(i-1,j))/(2*dxi);
        y_xi = (y(i+1,j)-y(i-1,j))/(2*dxi);
        end


        if j == 1
        x_eta = (-3*x(i,j)+4*x(i,j+1)-x(i,j+2))/(2*deta);
        y_eta = (-3*y(i,j)+4*y(i,j+1)-y(i,j+2))/(2*deta);

        elseif j == JM
        x_eta = (3*x(i,j)-4*x(i,j-1)+x(i,j-2))/(2*deta);
        y_eta = (3*y(i,j)-4*y(i,j-1)+y(i,j-2))/(2*deta);

        else
        x_eta = (x(i,j+1)-x(i,j-1))/(2*deta);
        y_eta = (y(i,j+1)-y(i,j-1))/(2*deta);
        end
        inv = (x_xi*y_eta)-(y_xi*x_eta);
        J = 1/inv;
        xi_x(i,j) = J*y_eta;
        xi_y(i,j) = -J*x_eta;
        eta_x(i,j) = -J*y_xi;
        eta_y(i,j) = J*x_xi;

    end
end
figure(2)
plot(x,y,'-r',x',y','-r')
xlabel('x')
ylabel('y')



figure(3)
plot(xi,eta,'-r',xi',eta','-r')
xlabel('\xi')
ylabel('\eta')

figure(4)
plot3(xi,eta,xi_x,'-r',xi',eta',xi_x','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\xi_{x}')

figure(5)
plot3(xi,eta,xi_y,'-r',xi',eta',xi_y','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\xi_{y}')

figure(6)
plot3(xi,eta,eta_x,'-r',xi',eta',eta_x','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\eta_{x}')

figure(7)
plot3(xi,eta,eta_y,'-r',xi',eta',eta_y','-r')
```

```
xlabel('\xi')
ylabel('\eta')
zlabel('\eta_{y}')
view(122,48)
toc
```

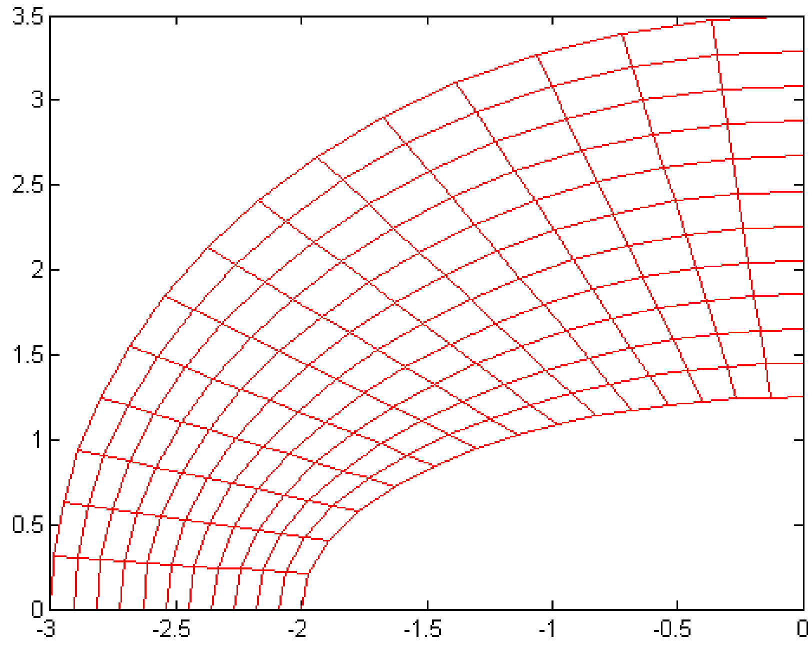


Figure 9.24 Algebraic grid system for the simplyconnected domain

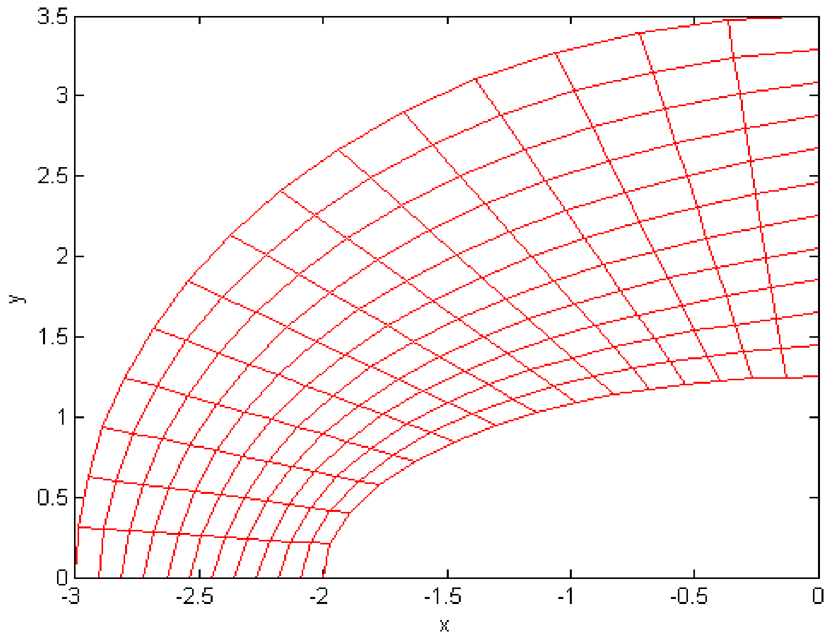
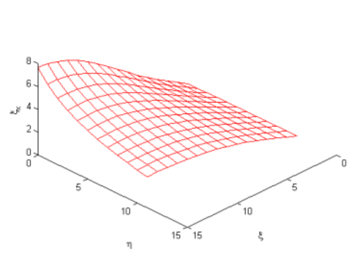
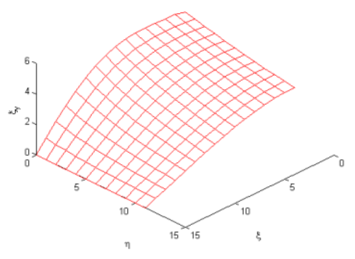


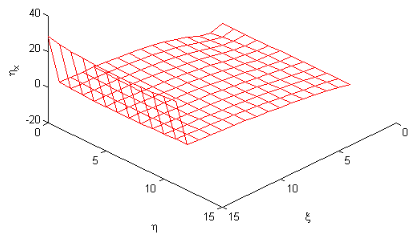
Figure 9.25 Grid system obtained by solving a system of elliptic partial differential equations



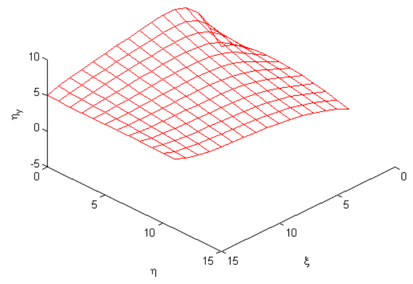
(a)



(b)



(c)



(d)

Figure 9.26a, b, c,d. Metric distributions for the grid system shown in Fig 9.25.

Doubly-Connected Domain

A doubly-connected domain is defined as a region which is not reducible. A domain which includes one configuration within the region of interest is classified as a doubly-connected domain. A doubly-connected region may be rendered simply-connected by introducing a suitable branch cut. This procedure is accomplished by inserting a branch cut that extends from a point on the body (interior boundary) to a point on the outer boundary. As an example, consider the airfoil shown in Figure 9.27. Select an outer boundary by specifying some geometrical configuration such as a circle, ellipse, rectangle, etc. In order to unwrap the domain such that a rectangular computational domain can be created, a branch cut, shown as line AC in Figure 9.27, is introduced. An intermediate step is shown in Figure 9.28.

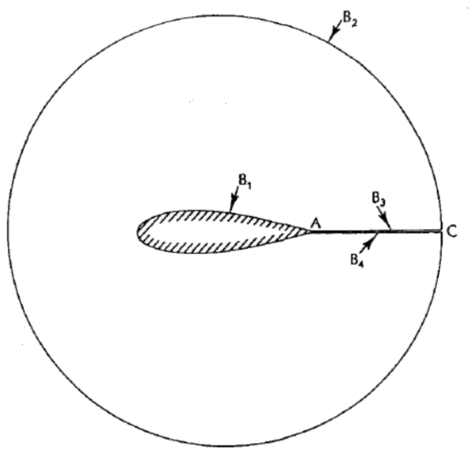


Figure 9.27 Doubly-connected region and the branch cut

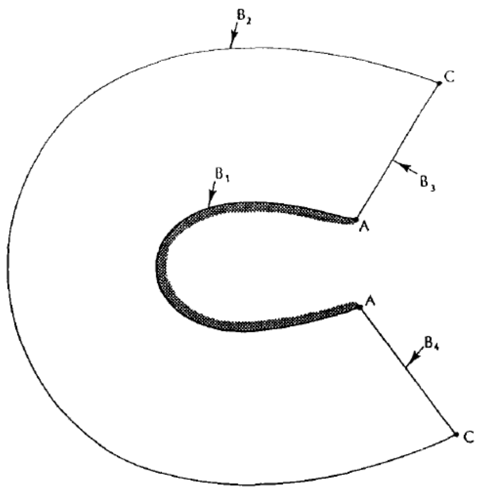


Figure 9.28 Unwrapping of the doubly-connected region

The domain is stretched and deformed to create a rectangular shape (computational domain) as shown in Figure 9.29.

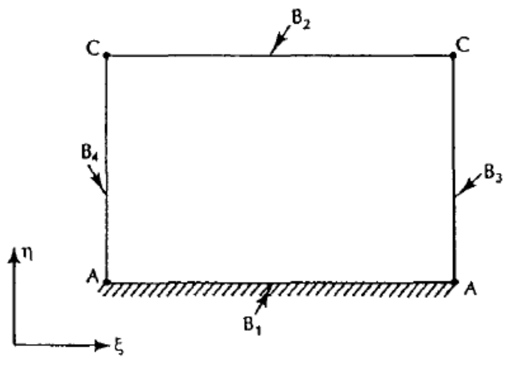


Figure 9.29 Computational domain

The boundaries of the domain are identified by B1, B2, B3, and B4. A uniformly distributed grid system is constructed in the computational space; therefore, the location of every grid point in the computational domain, including the boundaries, is known. The object then is to employ the elliptic grid generator to determine the location of grid points in the physical space. Thus, the elliptic equations to be solved are Equations (9.63) and (9.64). The procedure is similar to the one used for simply-connected regions. The grid point distribution on the boundaries of the physical domain is defined and an initial grid point distribution for the interior region is provided. One distinct difference is the treatment of grid points on the boundaries B_3 and B_4 , i.e., on the branch cut. These points must be free to float, i.e., the location of the grids along line AC must be updated. This update is accomplished by computing new values of $x_{1,J}$ and $y_{1,J}$ after each iteration. It is not necessary to compute the values of $x_{IM,J}$ and $y_{IM,J}$, since grid lines $i = 1$ and $i = IM$ are coincident; therefore, $x_{IM,J} = x_{1,J}$ and $y_{IM,J} = y_{1,J}$ · For the Gauss-Seidel formulation, $x_{1,J}$ and $y_{1,J}$ are computed according to

$$x_{i,j} = \left\{ \frac{a}{(\Delta\xi)^2}\left[x_{2,j} + x_{IM1,j}\right] + \frac{c}{(\Delta\eta)^2}\left[x_{1,j+1} + x_{1,j-1}\right] - \frac{b}{2\Delta\xi\Delta\eta}\left[x_{2,j+1} - x_{2,j-1} + x_{IM1,j-1} - x_{IM1,j+1}\right] / 2\left[\frac{a}{(\Delta\xi)^2} + \frac{c}{(\Delta\eta)^2}\right]\right\} \tag{9.76}$$

$$y_{i,j} = \left\{ \frac{a}{(\Delta\xi)^2}\left[y_{2,j} + y_{IM1,j}\right] + \frac{c}{(\Delta\eta)^2}\left[y_{1,j+1} + y_{1,j-1}\right] - \frac{b}{2\Delta\xi\Delta\eta}\left[y_{2,j+1} - y_{2,j-1} + y_{IM1,j-1} - y_{IM1,j+1}\right] / 2\left[\frac{a}{(\Delta\xi)^2} + \frac{c}{(\Delta\eta)^2}\right]\right\} \tag{9.77}$$

where $IM1 = IM - 1$.

Pseudo code :

```
%     Update x(1,j),y(1,j) and x(IM,j),y(IM,j)
%     computation of coefficients a,b & c lag by one iterative level

 for i = 1
        for j = 2:JM-1

    a = ((x(i,j+1)-x(i,j-1))/(2*deta))^2+((y(i,j+1)-y(i,j-1))/(2*deta))^2;
    b = ((x(i+1,j)-x(IM-1,j))/(2*dxi))*((x(i,j+1)-x(i,j-
1))/(2*deta))+((y(i+1,j)-y(IM-1,j))/(2*dxi))*((y(i,j+1)-y(i,j-1))/(2*deta));
    c = ((x(i+1,j)-x(IM-1,j))/(2*deta))^2+((y(i+1,j)-y(IM-1,j))/(2*deta))^2;
     a_1(i,j) = a/(dxi)^2;
     b_1(i,j) = b/(2*dxi*deta);
     c_1(i,j) = c/(deta)^2;
        end
    end
for i = 1
    for j = 2:JM-1
        ax_2 = a_1(i,j)*(x(i+1,j)+x(IM-1,j));
        bx_2 = b_1(i,j)*(x(i+1,j+1)-x(i+1,j-1)+x(IM-1,j-1)-x(IM-1,j+1));
        cx_2 = c_1(i,j)*(x(i,j+1)+x(i,j-1));
        ay_2 = a_1(i,j)*(y(i+1,j)+y(IM-1,j));
        by_2 = b_1(i,j)*(y(i+1,j+1)-y(i+1,j-1)+y(IM-1,j-1)-y(IM-1,j+1));
        cy_2 = c_1(i,j)*(y(i,j+1)+y(i,j-1));
        x(i,j) = (ax_2+cx_2-bx_2)/(2*(a_1(i,j)+c_1(i,j)));
        y(i,j) = (ay_2+cy_2-by_2)/(2*(a_1(i,j)+c_1(i,j)));
    end
end
```

These equations are used after each iteration to determine the new location of the grid points on the branch cut. Note that if the grid points on the branch cut are kept fixed, highly skewed grids at the branch cut are produced, which is undesirable.

To illustrate, we have used NACA four digit series airfoil generator code so that we can generate grids around any airfoil. The *x*-coordinates and *y*-coordinates of the grid points on the body can be defined as *x(i, 1)* and *y(i,1)* around the airfoil. The grid points on the outer boundary are evaluated as follows. The origin of the is placed at mid-chord and use equally spaced angular positions for each point on the outer boundary.

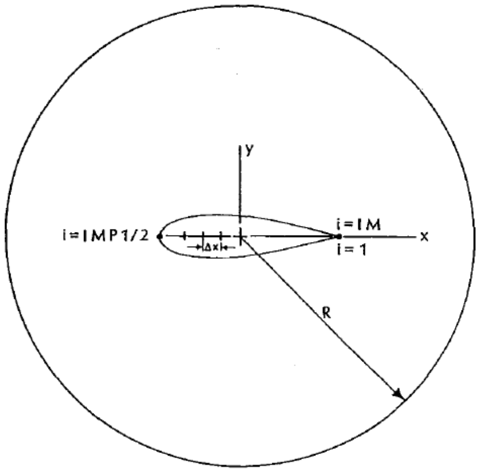


Figure 9.30 Physical domain used to illustrate the application of an elliptic grid generator to a doubly-connected region

The corresponding grid points for each *i* on the body (where *j* = 1) and on the outer boundary (where *j* = *J M*) are connected by straight lines. The grid points in the interior region are distributed along these lines. This distribution may be equally spaced, or some sort of grid clustering scheme employed. Thus, an algebraic grid system is constructed. Then the elliptic PDEs given by Equations (9.63) and (9.64) is solved.

The matlab code to generate physical domain and its metrics is given below:

(Doubly_Connected_Domain.m)

```
tic
clc
clear all
close all

c = 1.0;
IM = 39;
JM = 21;
R = 2;




%Airfoil coordinates
nacaseries = input('Enter the 4-digit naca series = ','s');


 % creating points on airfoil

 s1 = str2double(nacaseries(1));
 s2 = str2double(nacaseries(2));
 s3 = str2double(nacaseries(3));
 s4 = str2double(nacaseries(4));
  m = s1*0.01; p = s2*0.1 ; t = (10*s3+s4)*0.01;
 n = IM-1;

for i= n:-1:1

    theta = (i-n)*2*pi/n;
     x(n+1-i,JM) = R*cos(theta);
     y(n+1-i,JM) = R*sin(theta);
    xc = 0.5*c*(1+cos(theta));
if(xc/c)<p
    yc(n+1-i) = m*c/p^2*(2*p*(xc/c)-(xc/c)^2);
    dydx(n+1-i) = (2*m/p^2)* (p-xc/c);
    beta(n+1-i) = atan(dydx(n+1-i));
else
    yc(n+1-i) = m*c/(1-p)^2 * ((1-2*p)+2*p*(xc/c)-(xc/c)^2);
    dydx(n+1-i) = (2*m/(1-p)^2)* (p-xc/c);
    beta(n+1-i) = atan(dydx(n+1-i));
end
yt=5*t*c*(0.2969*sqrt(xc/c)-0.1260*(xc/c)...
    -0.3516*(xc/c)^2+0.2843*(xc/c)^3-0.1036*(xc/c)^4);

if(i<(0.5*n+1))
    xa(n+1-i)=xc - yt*sin(beta(n+1-i));
    ya(n+1-i)=yc(n+1-i)+yt*cos(beta(n+1-i));
else
    xa(n+1-i)=xc + yt*sin(beta(n+1-i));
    ya(n+1-i)=yc(n+1-i)-yt*cos(beta(n+1-i));
end

end
x(n+1,JM) = R;
y(n+1,JM) = 0;
xa(n+1)= c ;
```

```
ya(n+1) = 0;
yc(n+1) = 0;  % trailing edge

%Place midchord of the airfoil at the origin
for i = 1:IM
    x(i,1) = xa(i)-0.5;
    y(i,1) = ya(i);
end

dxi = 1.0;
deta = 1.0;

% computing ccoordinates of computational domain
for i = 1:IM
    for j = 1:JM
        xi(i,j) = (i-1)*dxi;
        eta(i,j) = (j-1)*deta;
    end
end

beta = 1.02;
beta1 = (beta+1)/(beta-1);


% Algebraic grid system generator

for i = 1:IM
    dx = x(i,JM)-x(i,1);
    dy = y(i,JM)-y(i,1);
    delta = sqrt(dx^2+dy^2);
    phi = atan2(dy,dx);
    for j = 1:JM
         gamma = (j-1)/(JM-1);
        num = 1-(beta1)^(1-gamma);
        den = 1+(beta1)^(1-gamma);
        beta2 = num/den;
        c = delta*(1+beta*beta2);
        x(i,j) = x(i,1)+c * cos(phi);
        y(i,j) = y(i,1)+c * sin(phi);
    end
end

figure(1)
plot(x,y,'-r',x',y','-r')
errormax = 0.01;
count = 1;
iteration = 0;


while count>0
    count = 0;
    prev_x = x;
    prev_y = y;

%     Update x(1,j),y(1,j) and x(IM,j),y(IM,j)
%     computation of coefficients a,b & c lag by one iterative level

 for i = 1
        for j = 2:JM-1

    a = ((x(i,j+1)-x(i,j-1))/(2*deta))^2+((y(i,j+1)-y(i,j-1))/(2*deta))^2;
    b = ((x(i+1,j)-x(IM-1,j))/(2*dxi))*((x(i,j+1)-x(i,j-
1))/(2*deta))+((y(i+1,j)-y(IM-1,j))/(2*dxi))*((y(i,j+1)-y(i,j-1))/(2*deta));
    c = ((x(i+1,j)-x(IM-1,j))/(2*deta))^2+((y(i+1,j)-y(IM-1,j))/(2*deta))^2;
     a_1(i,j) = a/(dxi)^2;
     b_1(i,j) = b/(2*dxi*deta);
     c_1(i,j) = c/(deta)^2;
        end
    end
for i = 1
    for j = 2:JM-1
        ax_2 = a_1(i,j)*(x(i+1,j)+x(IM-1,j));
        bx_2 = b_1(i,j)*(x(i+1,j+1)-x(i+1,j-1)+x(IM-1,j-1)-x(IM-1,j+1));
        cx_2 = c_1(i,j)*(x(i,j+1)+x(i,j-1));
        ay_2 = a_1(i,j)*(y(i+1,j)+y(IM-1,j));
        by_2 = b_1(i,j)*(y(i+1,j+1)-y(i+1,j-1)+y(IM-1,j-1)-y(IM-1,j+1));
        cy_2 = c_1(i,j)*(y(i,j+1)+y(i,j-1));
        x(i,j) = (ax_2+cx_2-bx_2)/(2*(a_1(i,j)+c_1(i,j)));
        y(i,j) = (ay_2+cy_2-by_2)/(2*(a_1(i,j)+c_1(i,j)));
    end
end
x(IM,:) = x(1,:);
y(IM,:) = y(1,:);
  %    Equation (9.68) & (9.69)
%     computation of coefficients a,b & c lag by one iterative level
    for i = 2:IM-1
        for j = 2:JM-1

     a = ((x(i,j+1)-x(i,j-1))/(2*deta))^2+((y(i,j+1)-y(i,j-1))/(2*deta))^2;
     b = ((x(i+1,j)-x(i-1,j))/(2*dxi))*((x(i,j+1)-x(i,j-
1))/(2*deta))+((y(i+1,j)-y(i-1,j))/(2*dxi))*((y(i,j+1)-y(i,j-1))/(2*deta));
     c = ((x(i+1,j)-x(i-1,j))/(2*dxi))^2+((y(i+1,j)-y(i-1,j))/(2*dxi))^2;
     a_1(i,j) = a/(dxi)^2;
     b_1(i,j) = b/(2*dxi*deta);
     c_1(i,j) = c/(deta)^2;
        end
    end




    for i = 2:IM-1
        for j = 2:JM-1
            ax_2 = a_1(i,j)*(x(i+1,j)+x(i-1,j));
            bx_2 = b_1(i,j)*(x(i+1,j+1)-x(i+1,j-1)+x(i-1,j-1)-x(i-1,j+1));
            cx_2 = c_1(i,j)*(x(i,j+1)+x(i,j-1));
            ay_2 = a_1(i,j)*(y(i+1,j)+y(i-1,j));
            by_2 = b_1(i,j)*(y(i+1,j+1)-y(i+1,j-1)+y(i-1,j-1)-y(i-1,j+1));
            cy_2 = c_1(i,j)*(y(i,j+1)+y(i,j-1));
            x(i,j) = (ax_2+cx_2-bx_2)/(2*(a_1(i,j)+c_1(i,j)));
            y(i,j) = (ay_2+cy_2-by_2)/(2*(a_1(i,j)+c_1(i,j)));
        end
    end

%     Error monitoring

    errorX = 0 ;
    errorY = 0 ;


    for i = 1:IM
        for j = 1:JM
            errorX = errorX + abs(x(i,j)-prev_x(i,j));
            errorY = errorY + abs(y(i,j)-prev_y(i,j));

        end
    end
    errorT = errorX+errorY;
    count = count+1;
    iteration = iteration+1,  errorT

    if (errorT<errormax)
        break;
    end
end


% computation of metrics
for i = 1:IM
    for j = 1:JM
        if i == 1
        x_xi(i,j) = (-3*x(i,j)+4*x(i+1,j)-x(i+2,j))/(2*dxi);
        y_xi(i,j) = (-3*y(i,j)+4*y(i+1,j)-y(i+2,j))/(2*dxi);

        elseif i == IM
        x_xi(i,j) = (3*x(i,j)-4*x(i-1,j)+x(i-2,j))/(2*dxi);
        y_xi(i,j) = (3*y(i,j)-4*y(i-1,j)+x(i-2,j))/(2*dxi);

        else
        x_xi(i,j) = (x(i+1,j)-x(i-1,j))/(2*dxi);
        y_xi(i,j) = (y(i+1,j)-y(i-1,j))/(2*dxi);
        end


        if j == 1
        x_eta(i,j) = (-3*x(i,j)+4*x(i,j+1)-x(i,j+2))/(2*deta);
        y_eta(i,j) = (-3*y(i,j)+4*y(i,j+1)-y(i,j+2))/(2*deta);

        elseif j == JM
        x_eta(i,j) = (3*x(i,j)-4*x(i,j-1)+x(i,j-2))/(2*deta);
        y_eta(i,j) = (3*y(i,j)-4*y(i,j-1)+y(i,j-2))/(2*deta);

        else
        x_eta(i,j) = (x(i,j+1)-x(i,j-1))/(2*deta);
        y_eta(i,j) = (y(i,j+1)-y(i,j-1))/(2*deta);
        end
    end
end




figure(2)
plot(x,y,'-r',x',y','-r')
xlabel('x')
ylabel('y')


figure(3)
plot(xi,eta,'-r',xi',eta','-r')
xlabel('\xi')
ylabel('\eta')

figure(4)
plot3(xi,eta,x_xi,'-r',xi',eta',x_xi','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('x_{\xi}')
view(139,26)

figure(5)
plot3(xi,eta,y_xi,'-r',xi',eta',y_xi','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('y_{\xi}')
view(139,26)

figure(6)
plot3(xi,eta,x_eta,'-r',xi',eta',x_eta','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('x_{\eta}')
view(139,26)

figure(7)
plot3(xi,eta,y_eta,'-r',xi',eta',y_eta','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('y_{\eta}')
view(139,26)
toc

% Author comment : y_xi has to be fixed...Others are perfect..
```

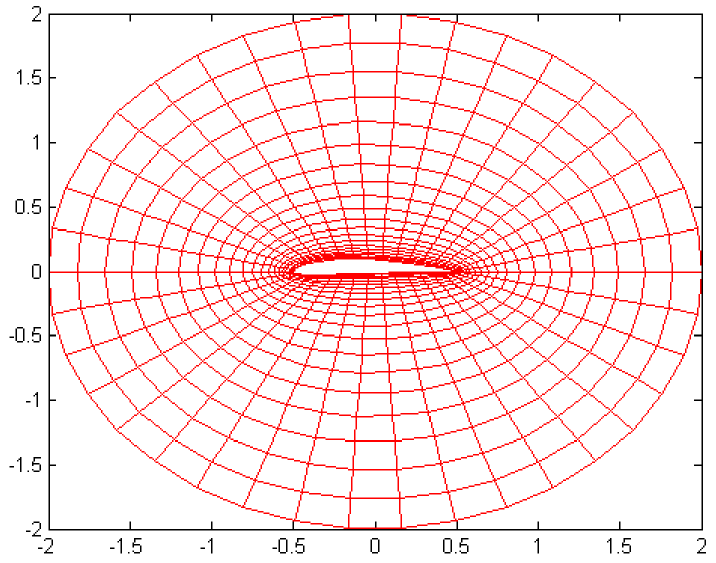


Figure 9.31 Algebraic grid system for the doubly-connected domain where interior boundary is NACA4412

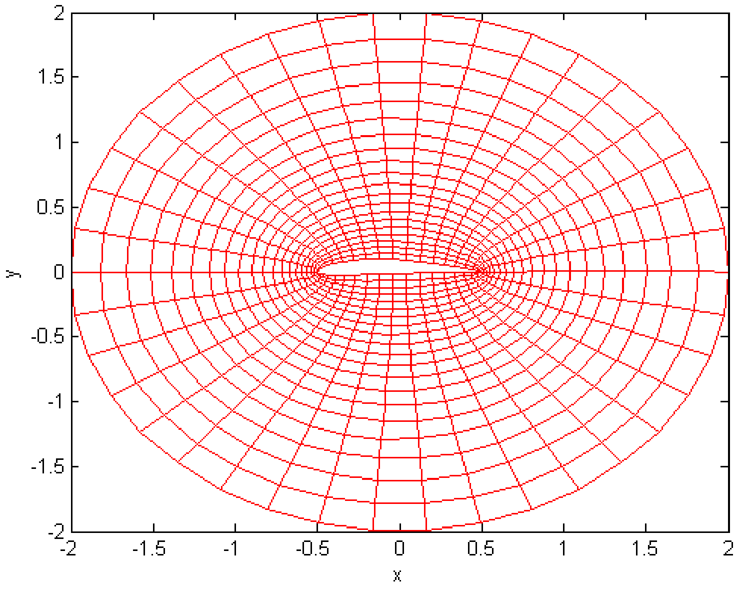
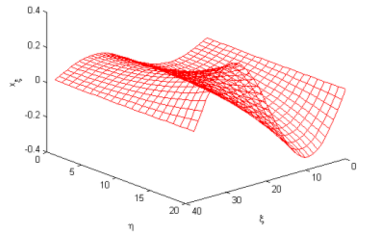
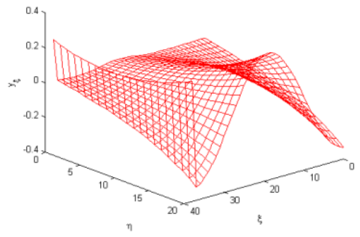


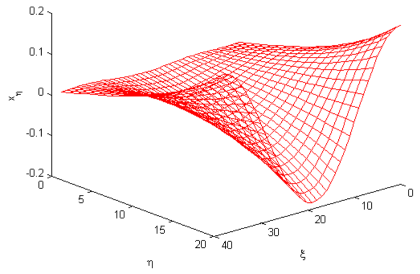
Figure 9.32 Grid system obtained by solving a system of elliptic partial differential equations



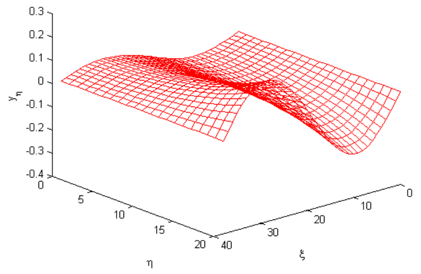
(a)



(b)



(c)



(d)

Figure 9.33a, b, c,d. Metric distributions for the grid system shown in Fig 9.32.

The figure above clearly indicates the smooth distribution of the grid points within the domain. We can see the adjustment of the grid points on the branch cut, which was used to define the two boundaries (B_3 and B_4 shown in Figure 9.27). This type grid is known as the "O" grid.

The selection of the physical domain is dependent on the particular fluid mechanics problem to be solved. The method by which the initial grid distribution is created is arbitrary and any procedure can be used. Before solving the differential equations of fluid motion, the grid system must be investigated carefully for grid point smoothness, skewness, orthogonality, and the metric distributions. The distributions of the transformation derivatives are shown in Figures 9.33a through 9.33d, which indicate well-behaved distributions.

Multiply-Connected Domain

The same procedure which is applied to a doubly-connected region can be extended to a multiply-connected region. For a multiply-connected region, more than one object is located within the domain. A branch cut is introduced to connect one body to the outer boundary. In addition, other cuts are inserted between various objects within the domain. A graphical illustration is shown in Figure 9.34 which represents the physical domain.

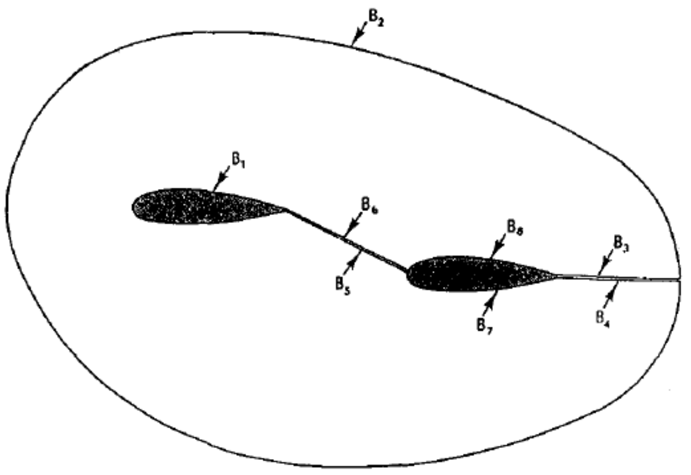


Figure 9.34 Physical domain for a multiply connected region

Figure 9.35 shows a typical intermediate step, and Figure 9.36 represents the computational space.

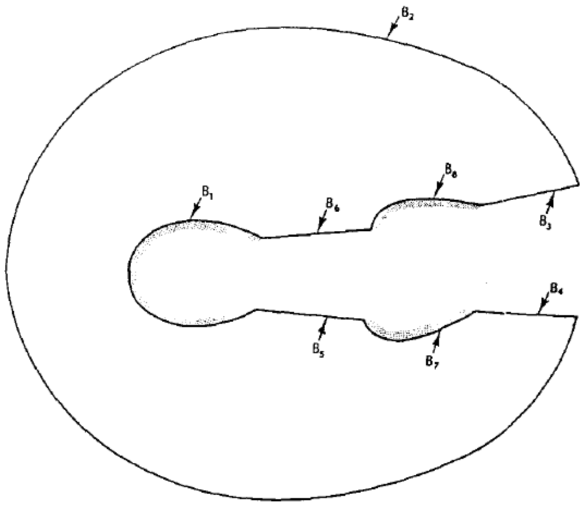


Figure 9.35 Unwrapping of a multiply connected region

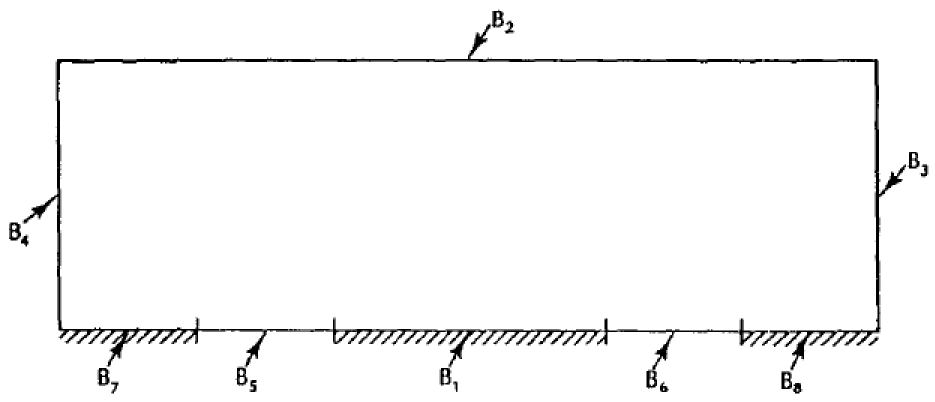


Figure 9.36 Computational domain

The elliptic equations (9.63) and (9.64) are solved by any iterative method for this domain. The procedure for specifying the boundary points and initial distribution of the grid points within the domain is similar to that of a doubly connected region. Again, the grid points on the branch cuts is computed after each iteration level, i.e., the grid points on the branch cuts and, thus, the shape of the branch cuts, are changing from one iteration to the next. The iterative procedure is continued until a specified convergence criterion is satisfied. For illustrative purposes, consider the airfoil described in the previous application. Now, insert two such airfoils within the circular domain. Grid points are distributed on the bodies and the outer boundary as well as the branch cuts. Once the grid points on the boundaries are specified, the grid point distribution within the domain can be determined by any algebraic procedure.

The matlab code to generate physical domain and its metrics is given below:

(Multiply_Connected_Domain.m)

```
tic
clc
clear all
close all

c = 1.0;
IM1 = 41;                    % Number of nodes on first airfoil
IM2 = 31;                    % Number of nodes on the line joining first &
second airfoil
IM3 = IM1;                   % Number of nodes on second airfoil
IM = IM1+2*IM2+IM3+1;        % Total nodes
JM = 21;
R = 4;

%Airfoil coordinates
```

```
nacaseries = input('Enter the 4-digit naca series = ','s');


 % creating points on airfoil

 s1 = str2double(nacaseries(1));
 s2 = str2double(nacaseries(2));
 s3 = str2double(nacaseries(3));
 s4 = str2double(nacaseries(4));
  m = s1*0.01; p = s2*0.1 ; t = (10*s3+s4)*0.01;
 n = IM1-1;

for i= 1:n

    theta = (i-1)*2*pi/n;
    xc = 0.5*c*(1+cos(theta));
if(xc/c)<p
    yc(i) = m*c/p^2*(2*p*(xc/c)-(xc/c)^2);
    dydx(i) = (2*m/p^2)* (p-xc/c);
    beta(i) = atan(dydx(i));
else
    yc(i) = m*c/(1-p)^2 * ((1-2*p)+2*p*(xc/c)-(xc/c)^2);
    dydx(i) = (2*m/(1-p)^2)* (p-xc/c);
    beta(i) = atan(dydx(i));
end
yt=5*t*c*(0.2969*sqrt(xc/c)-0.1260*(xc/c)...
    -0.3516*(xc/c)^2+0.2843*(xc/c)^3-0.1036*(xc/c)^4);

if(i<(0.5*n+1))
    xa(i)=xc - yt*sin(beta(i));
    ya(i)=yc(i)+yt*cos(beta(i));
else
    xa(i)=xc + yt*sin(beta(i));
    ya(i)=yc(i)-yt*cos(beta(i));
end

end
xa(n+1)= c ;
ya(n+1) = 0;
yc(n+1) = 0;  % trailing edge
figure(1)
plot(xa,ya,'-*r')



% place first airfoil
for i = 1:IM1
    x1(i) = xa(i);
    y1(i) = ya(i);
end

% place second airfoil
m = 0.5*(IM3-1);
for i = 1:IM3
    if i<m+1
    x3(i) = xa(i+m)+(c+0.5);
    y3(i) = ya(i+m)-2*max(ya);

    elseif i == IM3
    x3(i) = xa(m+1)+(c+0.5);
    y3(i) = ya(m+1)-2*max(ya);
    else
     x3(i) = xa(i-m)+(c+0.5);
    y3(i) = ya(i-m)-2*max(ya);
    end
end

lx = x3(1)-x1(IM1);
ly = y3(1)-y1(IM1);

dxx = lx/(IM2+1);
dyy = ly/(IM2+1);

%coordinates of line joining two airfoils
for i = 1:IM2
    x2(i) = x1(IM1)+i*dxx;
    y2(i) = y1(IM1)+i*dyy;
end

% compute centre of the outer boundary

im2_half = (IM2-1)*0.5;
xmid = x2(im2_half);
ymid = y2(im2_half);


for i = 1:IM
   if i<=im2_half
    x(i,1) = x2(im2_half+1-i);
    y(i,1) = y2(im2_half+1-i);
   elseif i>im2_half && i<=(im2_half+IM1)
       x(i,1) = x1(i-im2_half);
       y(i,1) = y1(i-im2_half);
   elseif i>(im2_half+IM1) && i<=(im2_half+IM1+IM2)
       x(i,1) = x2(i-im2_half-IM1);
       y(i,1) = y2(i-im2_half-IM1);
   elseif i>(im2_half+IM1+IM2) && i<=(im2_half+IM1+IM2+IM3)
       x(i,1) = x3(i-im2_half-IM1-IM2);
       y(i,1) = y3(i-im2_half-IM1-IM2);
   else
       x(i,1) = x2((im2_half+IM1+IM2+IM3)+(IM2+1)-i);
       y(i,1) = y2((im2_half+IM1+IM2+IM3)+(IM2+1)-i);
   end
end
% Creating outer circular boundary

phi = linspace(pi/2,5*pi/2,IM);

for i = 1:IM
    x(i,JM) = xmid+R*cos(phi(i));
    y(i,JM) = ymid+R*sin(phi(i));
```

```
end

dxi = 1.0;
deta = 1.0;

% compute coordinates of computaional domain
for i = 1:IM
    for j = 1:JM
        xi(i,j) = (i-1)*dxi;
        eta(i,j) = (j-1)*deta;
    end
end

beta = 1.02;
beta1 = (beta+1)/(beta-1);

% Algebraic grid system generator
for i = 1:IM
    dx = x(i,JM)-x(i,1);
    dy = y(i,JM)-y(i,1);
    delta = sqrt(dx^2+dy^2);
    phi = atan2(dy,dx);
    for j = 1:JM
         gamma = (j-1)/(JM-1);
        num = 1-(beta1)^(1-gamma);
        den = 1+(beta1)^(1-gamma);
        beta2 = num/den;
        c = delta*(1+beta*beta2);
        x(i,j) = x(i,1)+c * cos(phi);
        y(i,j) = y(i,1)+c * sin(phi);
    end
end

figure(2)
plot(x,y,'-r')
hold on
plot(x(:,JM),y(:,JM),'-r',xmid,ymid,'*')

figure(3)
plot(x,y,'-r',x',y','-r')
errormax = 0.01;
count = 1;
iteration = 0;

% compute grid by elliptic PDE scheme

while count>0
    count = 0;
    prev_x = x;
    prev_y = y;

%     Update x(1,j),y(1,j) and x(IM,j),y(IM,j)
%     computation of coefficients a,b & c lag by one iterative level

 for i = 1
```

```
        for j = 2:JM-1

    a = ((x(i,j+1)-x(i,j-1))/(2*deta))^2+((y(i,j+1)-y(i,j-1))/(2*deta))^2;
    b = ((x(i+1,j)-x(IM-1,j))/(2*dxi))*((x(i,j+1)-x(i,j-
1))/(2*deta))+((y(i+1,j)-y(IM-1,j))/(2*dxi))*((y(i,j+1)-y(i,j-1))/(2*deta));
    c = ((x(i+1,j)-x(IM-1,j))/(2*deta))^2+((y(i+1,j)-y(IM-1,j))/(2*deta))^2;
     a_1(i,j) = a/(dxi)^2;
     b_1(i,j) = b/(2*dxi*deta);
     c_1(i,j) = c/(deta)^2;
        end
    end
for i = 1
    for j = 2:JM-1
        ax_2 = a_1(i,j)*(x(i+1,j)+x(IM-1,j));
        bx_2 = b_1(i,j)*(x(i+1,j+1)-x(i+1,j-1)+x(IM-1,j-1)-x(IM-1,j+1));
        cx_2 = c_1(i,j)*(x(i,j+1)+x(i,j-1));
        ay_2 = a_1(i,j)*(y(i+1,j)+y(IM-1,j));
        by_2 = b_1(i,j)*(y(i+1,j+1)-y(i+1,j-1)+y(IM-1,j-1)-y(IM-1,j+1));
        cy_2 = c_1(i,j)*(y(i,j+1)+y(i,j-1));
        x(i,j) = (ax_2+cx_2-bx_2)/(2*(a_1(i,j)+c_1(i,j)));
        y(i,j) = (ay_2+cy_2-by_2)/(2*(a_1(i,j)+c_1(i,j)));
    end
end
x(IM,:) = x(1,:);
y(IM,:) = y(1,:);
  %    Equation (9.68) & (9.69)
%     computation of coefficients a,b & c lag by one iterative level
    for i = 2:IM-1
        for j = 2:JM-1

     a = ((x(i,j+1)-x(i,j-1))/(2*deta))^2+((y(i,j+1)-y(i,j-1))/(2*deta))^2;
     b = ((x(i+1,j)-x(i-1,j))/(2*dxi))*((x(i,j+1)-x(i,j-
1))/(2*deta))+((y(i+1,j)-y(i-1,j))/(2*dxi))*((y(i,j+1)-y(i,j-1))/(2*deta));
     c = ((x(i+1,j)-x(i-1,j))/(2*dxi))^2+((y(i+1,j)-y(i-1,j))/(2*dxi))^2;
     a_1(i,j) = a/(dxi)^2;
     b_1(i,j) = b/(2*dxi*deta);
     c_1(i,j) = c/(deta)^2;
        end
    end



    for i = 2:IM-1
        for j = 2:JM-1
            ax_2 = a_1(i,j)*(x(i+1,j)+x(i-1,j));
            bx_2 = b_1(i,j)*(x(i+1,j+1)-x(i+1,j-1)+x(i-1,j-1)-x(i-1,j+1));
            cx_2 = c_1(i,j)*(x(i,j+1)+x(i,j-1));
            ay_2 = a_1(i,j)*(y(i+1,j)+y(i-1,j));
            by_2 = b_1(i,j)*(y(i+1,j+1)-y(i+1,j-1)+y(i-1,j-1)-y(i-1,j+1));
            cy_2 = c_1(i,j)*(y(i,j+1)+y(i,j-1));
            x(i,j) = (ax_2+cx_2-bx_2)/(2*(a_1(i,j)+c_1(i,j)));
            y(i,j) = (ay_2+cy_2-by_2)/(2*(a_1(i,j)+c_1(i,j)));
        end
    end

    errorX = 0 ;
```

```
        errorY = 0 ;


        for i = 1:IM
            for j = 1:JM
                errorX = errorX + abs(x(i,j)-prev_x(i,j));
                errorY = errorY + abs(y(i,j)-prev_y(i,j));

            end
        end
        errorT = errorX+errorY;
        count = count+1;
        iteration = iteration+1,  errorT

        if (errorT<errormax)
            break;
        end
    end



    for i = 1:IM
        for j = 1:JM
            if i == 1
            x_xi(i,j) = (-3*x(i,j)+4*x(i+1,j)-x(i+2,j))/(2*dxi);
            y_xi(i,j) = (-3*y(i,j)+4*y(i+1,j)-y(i+2,j))/(2*dxi);

            elseif i == IM
            x_xi(i,j) = (3*x(i,j)-4*x(i-1,j)+x(i-2,j))/(2*dxi);
            y_xi(i,j) = (3*y(i,j)-4*y(i-1,j)+x(i-2,j))/(2*dxi);

            else
            x_xi(i,j) = (x(i+1,j)-x(i-1,j))/(2*dxi);
            y_xi(i,j) = (y(i+1,j)-y(i-1,j))/(2*dxi);
            end


            if j == 1
            x_eta(i,j) = (-3*x(i,j)+4*x(i,j+1)-x(i,j+2))/(2*deta);
            y_eta(i,j) = (-3*y(i,j)+4*y(i,j+1)-y(i,j+2))/(2*deta);

            elseif j == JM
            x_eta(i,j) = (3*x(i,j)-4*x(i,j-1)+x(i,j-2))/(2*deta);
            y_eta(i,j) = (3*y(i,j)-4*y(i,j-1)+y(i,j-2))/(2*deta);

            else
            x_eta(i,j) = (x(i,j+1)-x(i,j-1))/(2*deta);
            y_eta(i,j) = (y(i,j+1)-y(i,j-1))/(2*deta);
            end

        end
    end
```

```
figure(4)
plot(x,y,'-r',x',y','-r')
xlabel('x')
ylabel('y')
%
%
figure(5)
plot(xi,eta,'-r',xi',eta','-r')
xlabel('\xi')
ylabel('\eta')
%
figure(6)
plot3(xi,eta,x_xi,'-r',xi',eta',x_xi','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('x_{\xi}')
view(139,26)

figure(7)
plot3(xi,eta,y_xi,'-r',xi',eta',y_xi','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('y_{\xi}')
view(139,26)

figure(8)
plot3(xi,eta,x_eta,'-r',xi',eta',x_eta','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('x_{\eta}')
view(139,26)

figure(9)
plot3(xi,eta,y_eta,'-r',xi',eta',y_eta','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('y_{\eta}')
view(139,26)
toc

% Author comment : y_xi has to be fixed...Others are perfect..
```

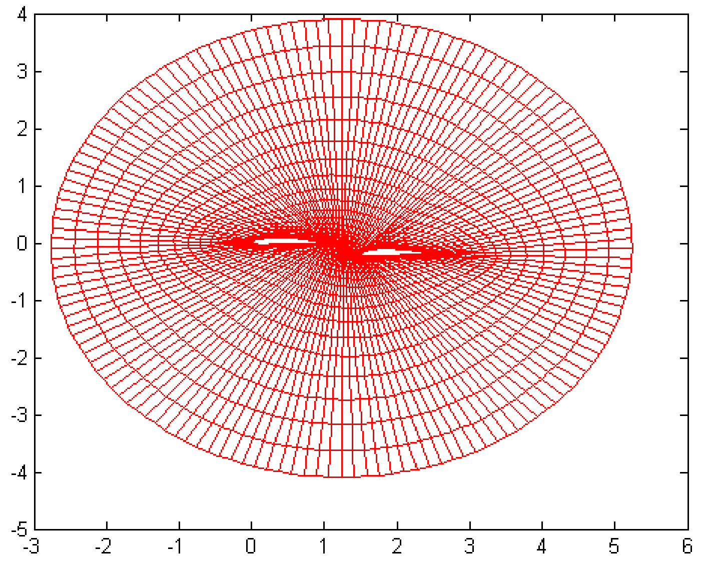


Figure 9.37 Algebraic grid system for the doubly-connected domain where interior boundary is NACA4412

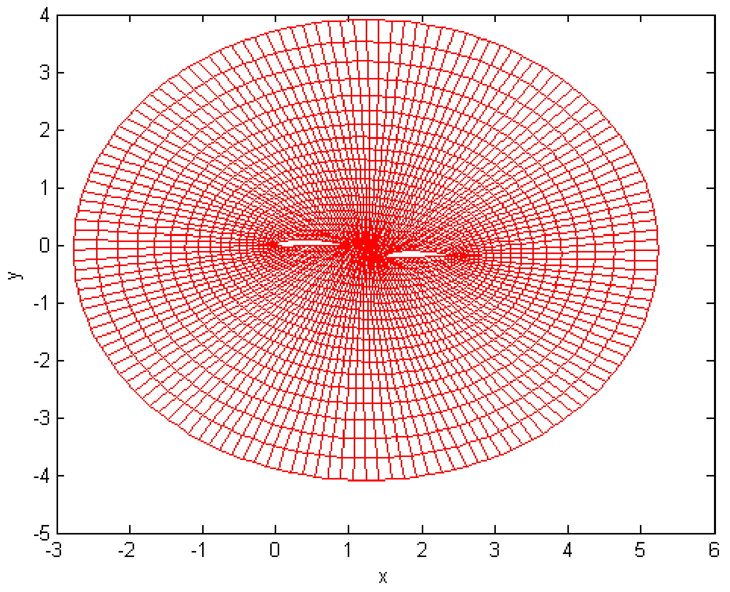


Figure 9.38 Grid system obtained by solving a system of elliptic partial differential equations

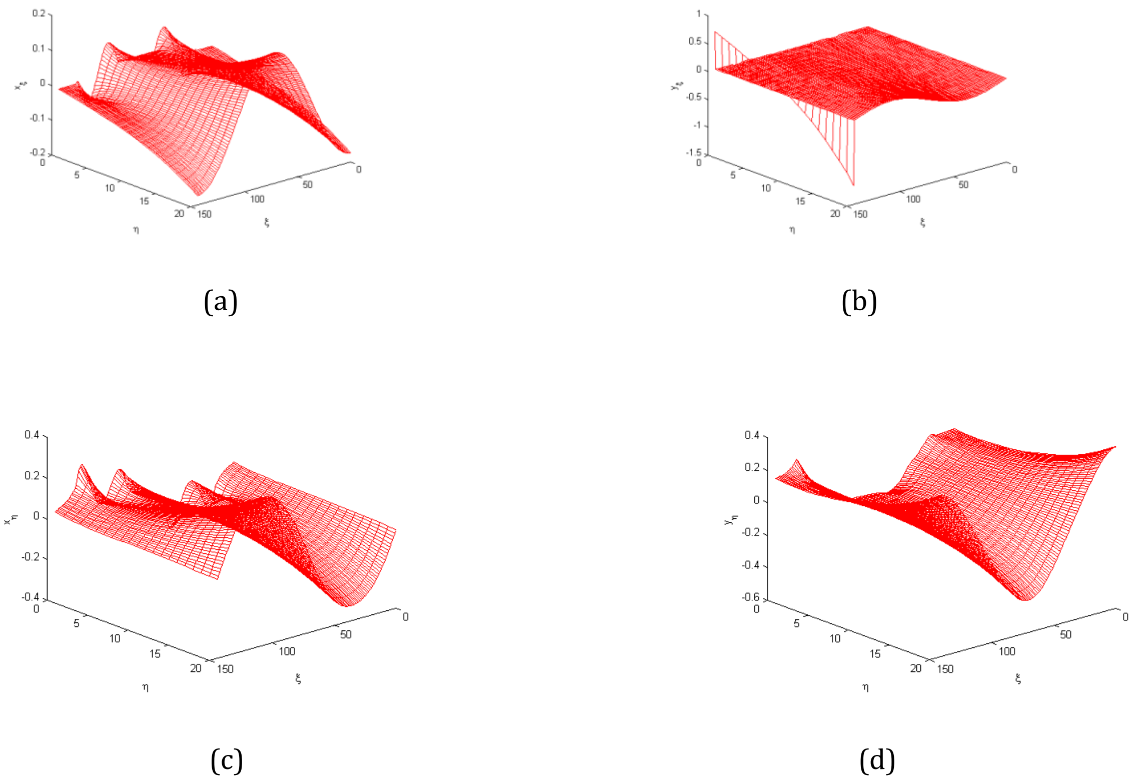


Figure 9.39a, b, c,d. Metric distributions for the grid system shown in Fig 9.38.

This method is used to generate the grid system shown in Figure 9.37. The crude distribution of grid points within the domain is clearly evident. This distribution is used as an initial condition to start the elliptic grid generator, i.e., Equations (9.63) and (9.64). The solution is shown in Figure 9.38. Two observations can be made. First, the grid points are distributed smoothly within the domain; second, the original branch cuts have been reshaped due to readjustment of the grid points.

C-Grid

An outer C-shaped domain is specified by a half circle and two parallel lines, as shown in Figure 9.40. Grid point distribution on the airfoil surface is to be clustered near the leading and trailing edges.

The following set of data is to be used:

$IT1 = 20$, $IL = 38$, L = 1.5 and $JM = 30$

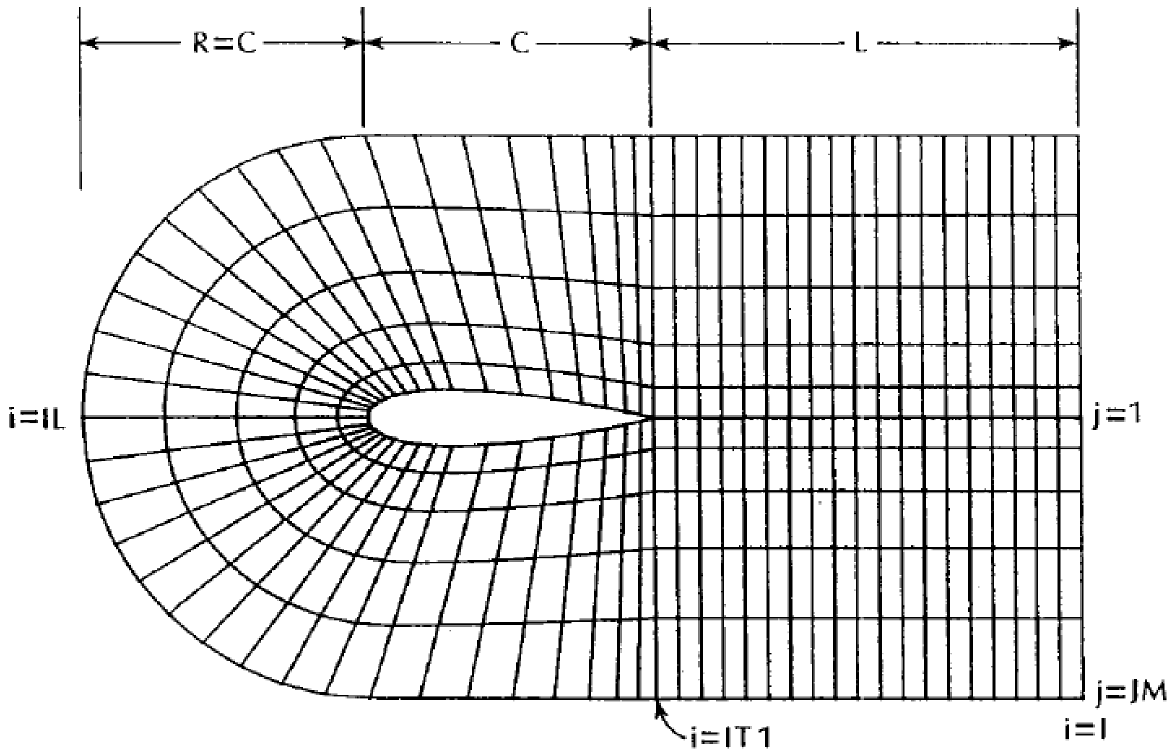


Figure 9.40 Nomenclature of given problem

The matlab code to generate physical domain is given below:

(c_grid.m)

```
tic
clc
clear all
close all

c = 1.0;
IT1 = 30;
```

```
IL = 48;
IM = 2*IL-1;
JM = 50;
R = c;
L = 1.5;
n = IM-2*(IT1-1)-1;                         % Number of nodes on airfoil
%Airfoil coordinates
nacaseries = input('Enter the 4-digit naca series = ','s');


 % creating points on airfoil

 s1 = str2double(nacaseries(1));
 s2 = str2double(nacaseries(2));
 s3 = str2double(nacaseries(3));
 s4 = str2double(nacaseries(4));
  m = s1*0.01; p = s2*0.1 ; t = (10*s3+s4)*0.01;



for i= 1:n

    theta = (i-1)*2*pi/n;
    xc = 0.5*c*(1+cos(theta));
if(xc/c)<p
    yc(i) = m*c/p^2*(2*p*(xc/c)-(xc/c)^2);
    dydx(i) = (2*m/p^2)* (p-xc/c);
    beta(i) = atan(dydx(i));
else
    yc(i) = m*c/(1-p)^2 * ((1-2*p)+2*p*(xc/c)-(xc/c)^2);
    dydx(i) = (2*m/(1-p)^2)* (p-xc/c);
    beta(i) = atan(dydx(i));
end
yt=5*t*c*(0.2969*sqrt(xc/c)-0.1260*(xc/c)...
    -0.3516*(xc/c)^2+0.2843*(xc/c)^3-0.1036*(xc/c)^4);

if(i<(0.5*n+1))
    xa(i)=xc - yt*sin(beta(i));
    ya(i)=yc(i)+yt*cos(beta(i));
else
    xa(i)=xc + yt*sin(beta(i));
    ya(i)=yc(i)-yt*cos(beta(i));
end

end
xa(n+1)= xa(1) ;
ya(n+1) = ya(1);
yc(n+1) = yc(1);  % trailing edge

figure(1)
plot(xa,ya,'*-r')

% compute x(i,1) starting from rear of physical domain
dx = L/(IT1-1);
aa = 0;
bb = 0;
```

```
cc = 0;
for i = 1:IM
    if i<IT1
        aa = aa+1;
        x(i,1) = xa(n+1)+dx*(IT1-i);
        y(i,1) = ya(n+1);
    elseif i>=IT1 && i<=IT1+n
        bb = bb+1;
        x(i,1) = xa(i+1-IT1);
        y(i,1) = ya(i+1-IT1);
    else
        cc = cc+1;
        x(i,1) = xa(n+1)+dx*(i-IT1-n);
        y(i,1) = ya(n+1);
    end

end

% Discretize outer boundary

I2 = 9;
c_m = IM-2*(I2+IT1-1);
phi = linspace(pi/2,3*pi/2,c_m);

dx1 = c/I2;
sc_n = IT1+I2-1+c_m;
for i = 1:IM
    if i<IT1
        x(i,JM) = x(i,1);
        y(i,JM) = y(i,1)+R;
    elseif i>=IT1 && i<=(IT1+I2-1)
        x(i,JM) =  dx1*(IT1+I2-i);
        y(i,JM) = ya(n+1)+R;
    elseif i>(IT1+I2-1) && i<=sc_n
        x(i,JM) = R*cos(phi(i-(IT1+I2-1)));
        y(i,JM) = R*sin(phi(i-(IT1+I2-1)));
    elseif i>sc_n && i<=(sc_n+I2)
        x(i,JM) = dx1*(i-sc_n);
        y(i,JM) = ya(n+1)-R;
    else
        x(i,JM) = c + dx*(i-(sc_n+I2));
        y(i,JM) = ya(n+1)-R;
    end
end

    figure(2)
    plot(x(:,1),y(:,1),x(:,JM),y(:,JM),'-r')

        dxi = 1.0;
        deta = 1.0;
        for i = 1:IM
            for j = 1:JM
                xi(i,j) = (i-1)*dxi;
                eta(i,j) = (j-1)*deta;
            end
        end
```

```
beta = 1.1;
beta1 = (beta+1)/(beta-1);

for i = 1:IM
    dx = x(i,JM)-x(i,1);
    dy = y(i,JM)-y(i,1);
    delta = sqrt(dx^2+dy^2);
    alpha = atan2(dy,dx);
    for j = 1:JM
        gamma = (j-1)/(JM-1);
        num = 1-(beta1)^(1-gamma);
        den = 1+(beta1)^(1-gamma);
        beta2 = num/den;
        cij = delta*(1+beta*beta2);
        x(i,j) = x(i,1)+cij * cos(alpha);
        y(i,j) = y(i,1)+cij * sin(alpha);
    end
end

figure(3)
plot(x,y,'-r',x',y','-r')
```

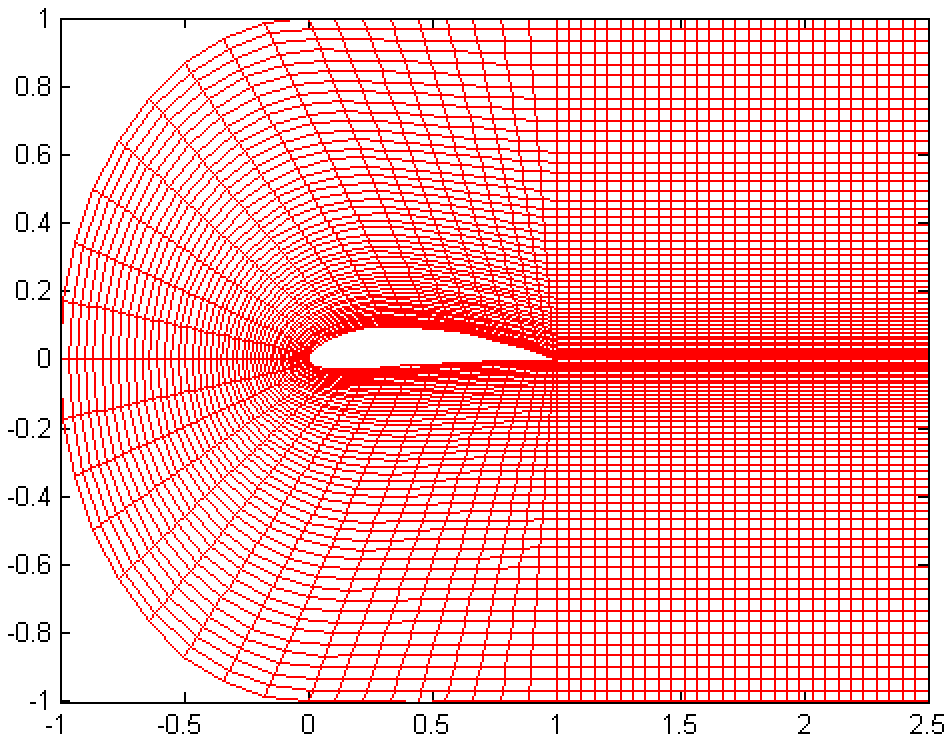


Figure 9.41 Algebraic c-grid system

P9.5 The domain shown in Figure 9.42 is defined by the following data: *A1* =3.0 , *A2* = 2.0 , *D* = 2.0 , *H* = 4.0. Generate a grid by an elliptic scheme. Use the following data: *I*1 = 10 , *IMID* = 14 , *I*2 = 18 , *IM* = 28 , and *JM* = 30.

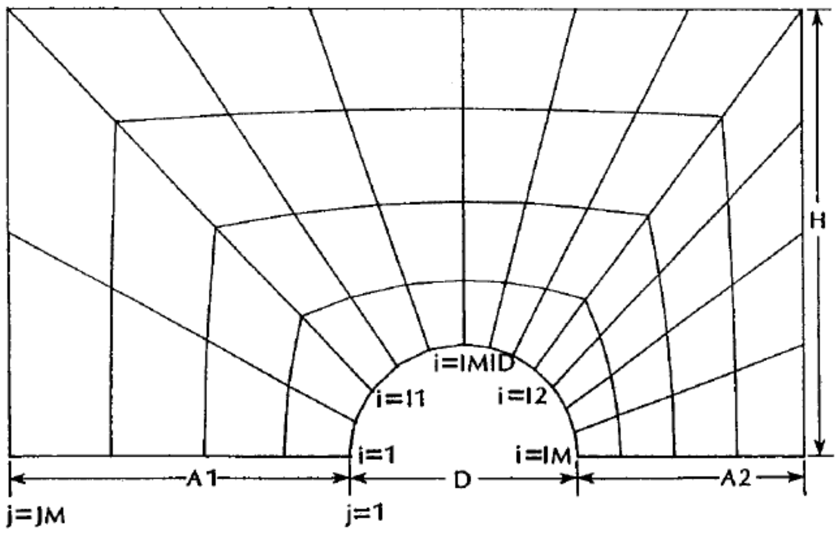


Figure 9.42 Nomenclature for the given problem

The matlab code to generate physical domain and its metrics is given below:

(simply_connected_cylinder.m)

```
clc
clear all
close all

A1 = 3.0;
A2 = 2.0;
D = 2.0;
R = D/2;
H = 4.0;
I1 = 10;
IMID = 14;
I2 = 18;
IM = 28;
JM = 30;

% Place the half cylinder at the distance of A1 from inlet

theta = linspace(pi,0,IM);

for i = 1:IM
    x(i,1) = A1+R+R*cos(theta(i));
    y(i,1) = R*sin(theta(i));
end
```

```
dy1 = H/(I1-1);
dy2 = H/(IM-I2);
dx = (A1+D+A2)/(I2-I1);

for i = 1:IM
    if i<=I1
        x(i,JM) = 0.0;
        y(i,JM) = (i-1)*dy1;
    elseif i>I1 && i<=I2
        x(i,JM) = (i-I1)*dx;
        y(i,JM) = H;
    else
        x(i,JM) = A1+D+A2;
        y(i,JM) = (IM-i)*dy2;
    end
end

dxi = 1.0;
deta = 1.0;
for i = 1:IM
    for j = 1:JM
        xi(i,j) = (i-1)*dxi;
        eta(i,j) = (j-1)*deta;
    end
end

beta = 1.07;
beta1 = (beta+1)/(beta-1);

for i = 1:IM
    dx = x(i,JM)-x(i,1);
    dy = y(i,JM)-y(i,1);
    delta = sqrt(dx^2+dy^2);
    phi = atan2(dy,dx);
    for j = 1:JM
         gamma = (j-1)/(JM-1);
        num = 1-(beta1)^(1-gamma);
        den = 1+(beta1)^(1-gamma);
        beta2 = num/den;
        c = delta*(1+beta*beta2);
        x(i,j) = x(i,1)+c * cos(phi);
        y(i,j) = y(i,1)+c * sin(phi);
    end
end

figure(1)
plot(x(:,1),y(:,1),'-r',x(:,JM),y(:,JM),'-r')
axis([0 A1+A2+D 0 H])

figure(2)
plot(x,y,'-r',x',y','-r')
axis([0 A1+A2+D 0 H])

errormax = 0.00001;
```

```
count = 1;
iteration = 0;

% compute grid by elliptic PDE scheme
while count>0
    count = 0;
    prev_x = x;
    prev_y = y;

%    Equation (9.68) & (9.69)
%     computation of coefficients a,b & c lag by one iterative level
    for i = 2:IM-1
        for j = 2:JM-1

     a = ((x(i,j+1)-x(i,j-1))/(2*deta))^2+((y(i,j+1)-y(i,j-1))/(2*deta))^2;
     b = ((x(i+1,j)-x(i-1,j))/(2*dxi))*((x(i,j+1)-x(i,j-
1))/(2*deta))+((y(i+1,j)-y(i-1,j))/(2*dxi))*((y(i,j+1)-y(i,j-1))/(2*deta));
     c = ((x(i+1,j)-x(i-1,j))/(2*dxi))^2+((y(i+1,j)-y(i-1,j))/(2*dxi))^2;
     a_1(i,j) = a/(dxi)^2;
     b_1(i,j) = b/(2*dxi*deta);
     c_1(i,j) = c/(deta)^2;
        end
    end




    for i = 2:IM-1
        for j = 2:JM-1
            ax_2 = a_1(i,j)*(x(i+1,j)+x(i-1,j));
            bx_2 = b_1(i,j)*(x(i+1,j+1)-x(i+1,j-1)+x(i-1,j-1)-x(i-1,j+1));
            cx_2 = c_1(i,j)*(x(i,j+1)+x(i,j-1));
            ay_2 = a_1(i,j)*(y(i+1,j)+y(i-1,j));
            by_2 = b_1(i,j)*(y(i+1,j+1)-y(i+1,j-1)+y(i-1,j-1)-y(i-1,j+1));
            cy_2 = c_1(i,j)*(y(i,j+1)+y(i,j-1));
            x(i,j) = (ax_2+cx_2-bx_2)/(2*(a_1(i,j)+c_1(i,j)));
            y(i,j) = (ay_2+cy_2-by_2)/(2*(a_1(i,j)+c_1(i,j)));
        end
    end

%     Error monitoring
    errorX = 0 ;
    errorY = 0 ;



    for i = 1:IM
        for j = 1:JM
            errorX = errorX + abs(x(i,j)-prev_x(i,j));
            errorY = errorY + abs(y(i,j)-prev_y(i,j));

        end
    end
    errorT = errorX+errorY;
    count = count+1;
    iteration = iteration+1,  errorT
    if (errorT<errormax)
        break;
    end
end

figure(3)
plot(x,y,'-r',x',y','-r')
axis([0 A1+A2+D 0 H])

for i = 1:IM
    for j = 1:JM
        if i == 1
        x_xi = (-3*x(i,j)+4*x(i+1,j)-x(i+2,j))/(2*dxi);
        y_xi = (-3*y(i,j)+4*y(i+1,j)-y(i+2,j))/(2*dxi);

        elseif i == IM
        x_xi = (3*x(i,j)-4*x(i-1,j)+x(i-2,j))/(2*dxi);
        y_xi = (3*y(i,j)-4*y(i-1,j)+x(i-2,j))/(2*dxi);

        else
        x_xi = (x(i+1,j)-x(i-1,j))/(2*dxi);
        y_xi = (y(i+1,j)-y(i-1,j))/(2*dxi);
        end


        if j == 1
        x_eta = (-3*x(i,j)+4*x(i,j+1)-x(i,j+2))/(2*deta);
        y_eta = (-3*y(i,j)+4*y(i,j+1)-y(i,j+2))/(2*deta);

        elseif j == JM
        x_eta = (3*x(i,j)-4*x(i,j-1)+x(i,j-2))/(2*deta);
        y_eta = (3*y(i,j)-4*y(i,j-1)+y(i,j-2))/(2*deta);

        else
        x_eta = (x(i,j+1)-x(i,j-1))/(2*deta);
        y_eta = (y(i,j+1)-y(i,j-1))/(2*deta);
        end
        inv = (x_xi*y_eta)-(y_xi*x_eta);
        J = 1/inv;
        xi_x(i,j) = J*y_eta;
        xi_y(i,j) = -J*x_eta;
        eta_x(i,j) = -J*y_xi;
        eta_y(i,j) = J*x_xi;

    end
end


figure(4)
plot(xi,eta,'-r',xi',eta','-r')
xlabel('\xi')
ylabel('\eta')

figure(5)
```

```
plot3(xi,eta,xi_x,'-r',xi',eta',xi_x','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\xi_{x}')
view(135,48)
figure(6)
plot3(xi,eta,xi_y,'-r',xi',eta',xi_y','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\xi_{y}')
view(135,48)
figure(7)
plot3(xi,eta,eta_x,'-r',xi',eta',eta_x','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\eta_{x}')
view(135,48)
figure(8)
plot3(xi,eta,eta_y,'-r',xi',eta',eta_y','-r')
xlabel('\xi')
ylabel('\eta')
zlabel('\eta_{y}')
view(135,48)
toc
```

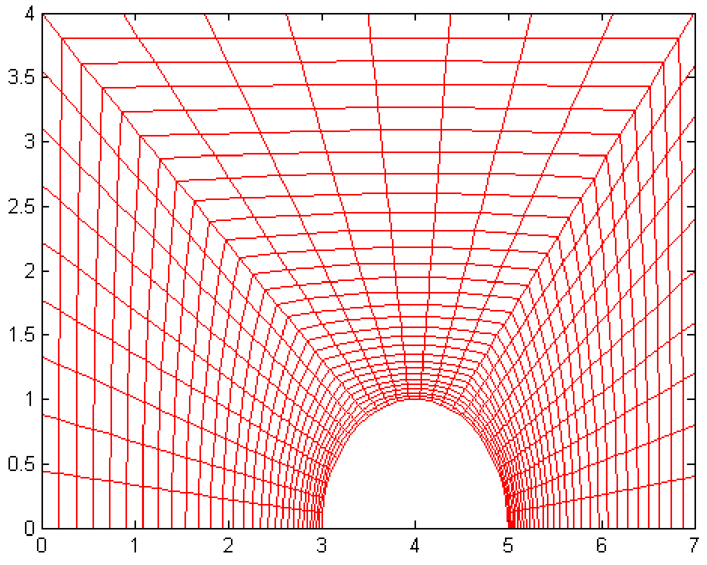


Figure 9.43 Algebraic grid system

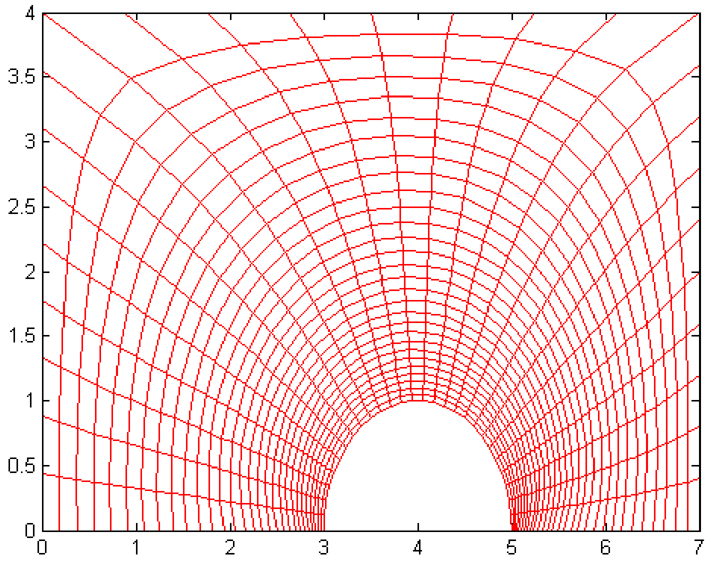


Figure 9.44 Grid system obtained by solving a system of elliptic partial differential equations

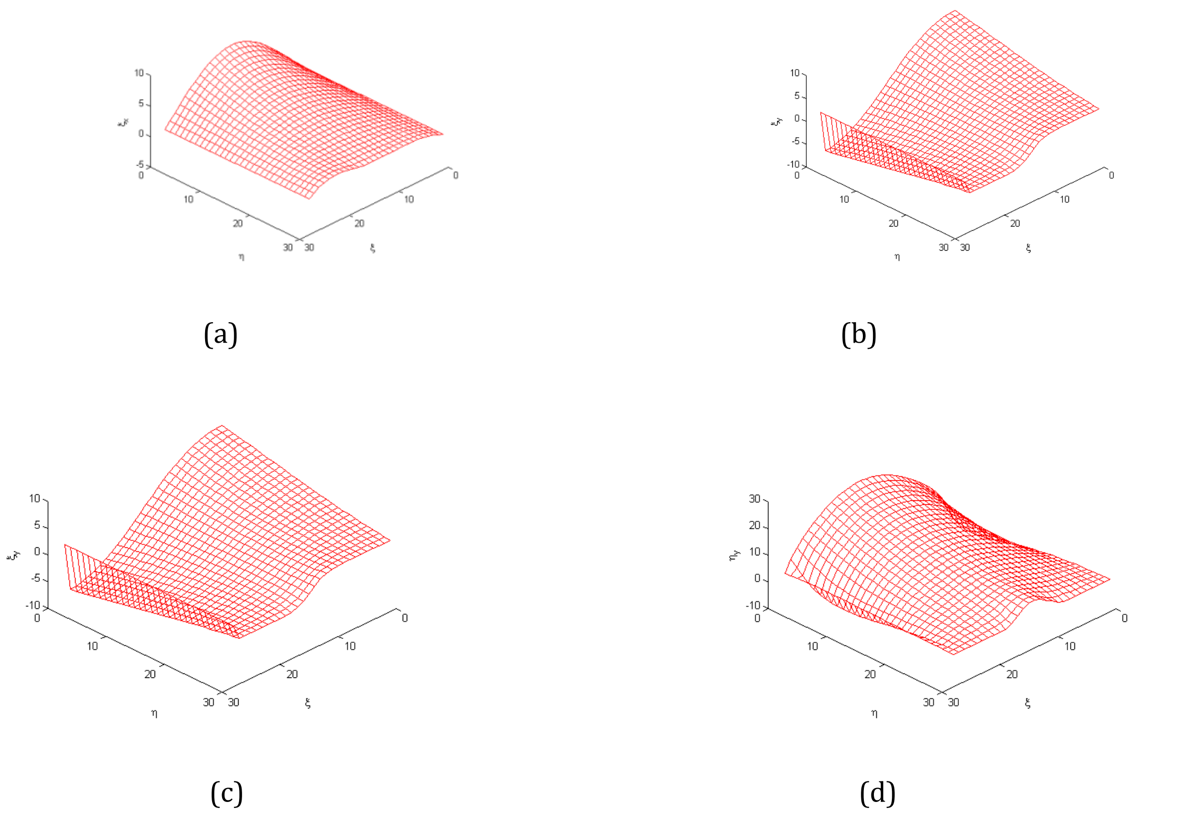


Figure 9.45a, b, c,d. Metric distributions for the grid system shown in Fig 9.44