Instituut
Wiskundige Dienstverlening
Eindhoven

REPORT IWDE 89-07

AN ANALYTIC MESH GENERATION METHOD BASED ON ELLIPTIC SYSTEMS

Maria Angela Petrilli

June 1989

AN ANALYTIC MESH GENERATION METHOD BASED ON ELLIPTIC SYSTEMS

Maria Angela Petrilli

Report IWDE 89-07
June 1989

The present work has been performed under contract with Zowel B.V. (Eindhoven) for the Instituut voor Wiskundige Dienstverlening Eindhoven. The author is a student of SASIAM/Tecnopolis (Bari, Italy). She visited the Eindhoven University of Technology from 1-4 to 30-6 1989 in the ECMI student exchange programme. IWDE supervisor was S.W. Rienstra (TUE).

ABSTRACT

The basic ideas of the construction and use of numerically generated boundary coordinate systems for numerical grid generation are discussed. With such coordinate systems, all computations can be done in a rectangular transformed region regardless of the shape of the physical boundaries. Furthermore, numerical solutions of partial differential equations on regions of arbitrary shape can be constructed, and codes can be developed, that treat different physical configurations and boundary shapes entirely as input. The method used to construct computational meshes will be based on the solution of an elliptic boundary value problem. Additional interior grid control is effected by a modified elliptic system containing "control functions" in two ways: either assigning the values of such control functions in input, or computing them from the Dirichlet boundary values by constraining the coordinate lines to be locally orthogonal to the boundary. The numerical algorithm is essentially a simple multigrid method, where the numbers of meshes in x and y direction have to be powers of two, allowing an iteration based on halving the meshsize.

1. INTRODUCTION

The motivation for grid generation is the numerical solution of partial differential equations by either finite difference or finite element techniques.

The way the domain is subdivided is very important since the computational mesh largely affects the stability and convergence of the numerical solution.

The present work has been done, primarily, to provide a mesh generator to be used with the finite-volume package "Phoenix" for internal (turbulent) heat and mass flow calculations.

Although the impetus for these developments has come from fluid dynamical problems, the general technique here illustrated is equally applicable to heat transfer, electromagnetics, solid mechanics, etc.

In the last years the numerical generation of boundary-conforming curvilinear coordinate systems has provided the key to the development of numerical grid generation techniques with arbitrarily shaped boundaries (see [3]).

A boundary-conforming coordinate system is a curvilinear coordinate system having some coordinate lines (surfaces in 3D) coincident with segments of the boundary of the region considered.

Boundary-conforming coordinate systems are generated numerically by determining the values of the physical coordinates in the field from the values (and/or angles of intersection) on the boundary. This can be done in two basic ways: by the Algebraic Generation Method and by the Analytic Generation Method.

The Algebraic Generation Method consists of an interpolation between the boundaries. Thus a general coordinate line is given as a function of the curvilinear coordinates. This function contains certain coefficients which are determined so that the function matches specified values of the cartesian coordinates. Algebraic Grid Generation is discussed in detail in [1].

In the Analytic Generation Method the curvilinear coordinates are defined via suitable partial differential equations producing, on the one hand, smooth coordinate lines and, on the other hand, allowing us, by the boundary conditions, to impose the prescribed values at the boundaries.

Although the solution of such a boundary-value problem is a classic problem of partial differential equations, some care is of course to be taken with respect to which equations are applicable and which solution method is appropriate.

In the following, an analytic method is presented for the generation of a general 2D mesh system.

2. ELLIPTIC GENERATION SYSTEMS

If the coordinate points are specified on the entire closed boundary of the physical region, the grid generating system must be elliptic, while if the specification is on only a suitable portion of the boundary, the system would be parabolic or hyperbolic. This latter case would occur, for instance, when an inner boundary of a physical region is specified, but a surrounding outer boundary is arbitrary.
The present work, however, treats the case of a completely specified boundary, which requires an elliptic partial differential system.

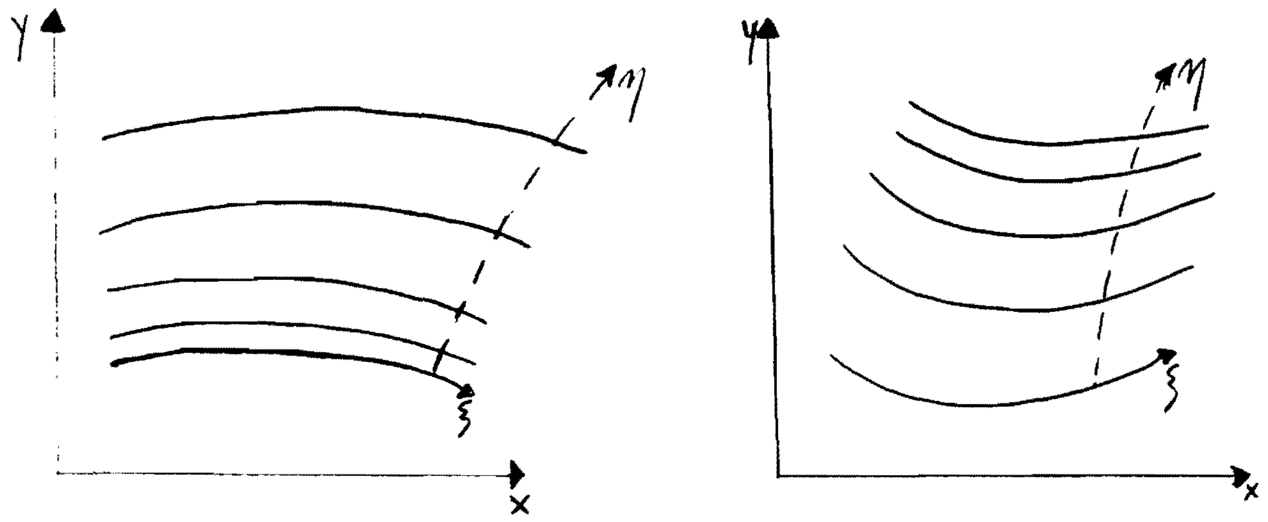
2.1. Laplace system

The most simple elliptic partial differential system is the Laplace system:

$$\nabla^2 \xi^i = 0 \quad i = 1,2 \tag{2.1}$$

where $\xi^1 = \xi$ and $\xi^2 = \eta$.

With this generating system the coordinate lines will tend to be equally spaced in the absence of boundary curvature because of the strong smoothing effect of the Laplacian, but will become more closely spaced over concave boundaries, and less so over convex boundaries, as illustrated below for a boundary $\eta = 0$.



$\eta_{xx} > 0$
(concave)

$\eta_{xx} < 0$
(convex)

In the left figure we have $\eta_{xx} > 0$ because of the concave curvature of the lines of constant η (= η-lines). Therefore it follows that $\eta_{yy} < 0$ and hence the spacing between the η-lines must increase with y. The η-lines thus will tend to be more closely spaced over such a concave boundary segment.

For convex segments, illustrated in the right figure, we have $\eta_{xx} < 0$, so that η_{yy} must be positive and hence the spacing of the η-lines must decrease outward from this convex boundary.

2.2. Poisson system

Control of the coordinate line distribution in the field can be exercised by generalizing the elliptic generating system (2.1) to Poisson equations:

$$\nabla^2 \xi^i = P^i \quad i = 1,2 \tag{2.2}$$

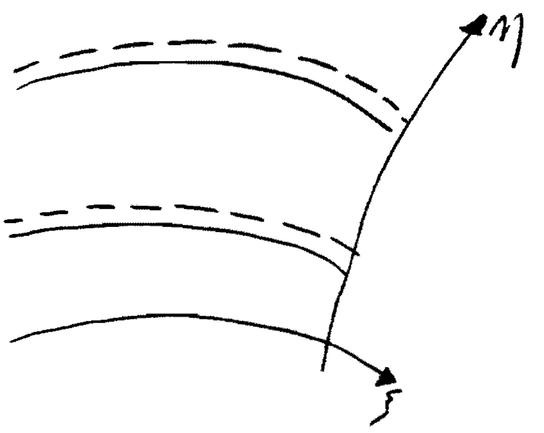
where $\xi^1 = \xi$, $\xi^2 = \eta$, $P^1 = P$, $P^2 = Q$.

In the system (2.2) the terms P^i can be fashioned to control the spacing of the coordinate lines and for this reason they are called "control functions". They may be considered to be dependent both on the dependent and independent variables.

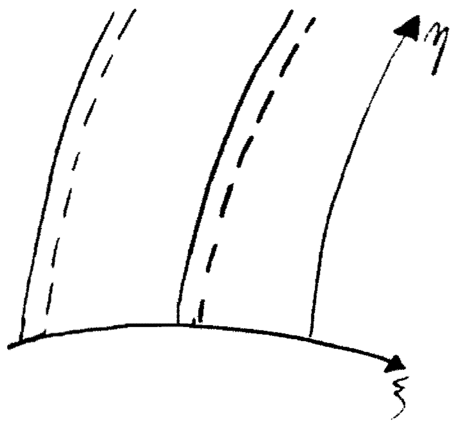
Considering the equation

$$\nabla^2 \xi = P ,$$

positive values of the control function P will cause the ξ-lines (= lines of constant ξ) to tend to move in the direction of increasing ξ and viceversa. Similarly for η.



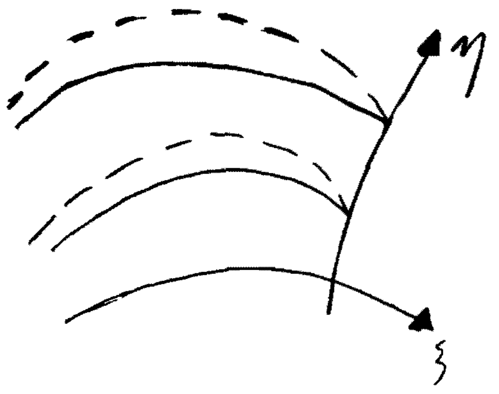
$Q > 0$

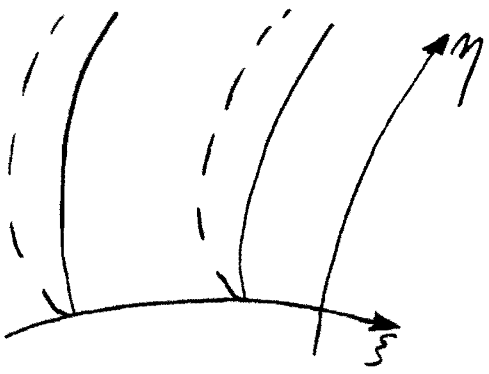


$P > 0$

With the boundary values fixed, as here considered, the η-lines and the ξ-lines cannot change the intersection with the boundary. So near the boundary the effect of the control functions P and Q is to change the angle of intersection at the boundary. For instance, negative values of P will cause the ξ-lines to lean in the direction of decreasing ξ and positive values of Q will cause the

η-lines to lean in the direction of increasing η.
These effects are illustrated in the following pictures:





In a word, the mesh lines can be dense or coarse depending on the negative or positive values of the control functions.

Various forms of the source terms *P* and *Q* have been devised that contain adjustable parameters but, however, the forms of these source terms and the values of the adjustable parameters require artful selection and are problem dependent.

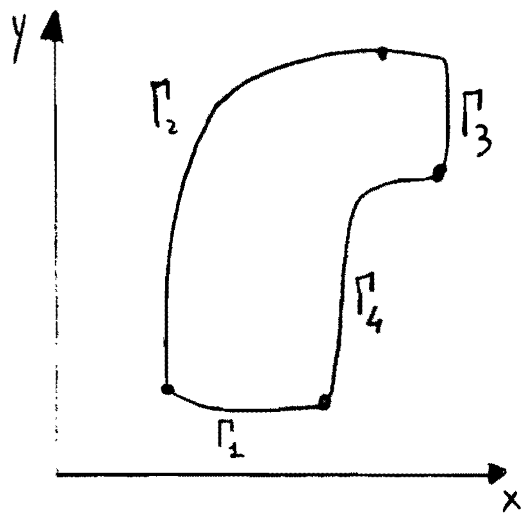
3. TRANSFORMED EQUATIONS

In order to make use of the Analytic Generation Method, the generating equations must first be transformed to the curvilinear coordinates.

Let us consider the Poisson system

$$\begin{cases} \dfrac{\partial^2 \xi}{\partial x^2} + \dfrac{\partial^2 \xi}{\partial y^2} = P & (3.1a) \\ \dfrac{\partial^2 \eta}{\partial x^2} + \dfrac{\partial^2 \eta}{\partial y^2} = Q & (3.1b) \end{cases}$$

with the curvilinear coordinates $\xi(x,y)$ and $\eta(x,y)$ specified on a boundary curve Γ.
Then the system (3.1) generates the values $\xi(x,y)$ and $\eta(x,y)$ in the field bounded by Γ. This is thus a boundary value problem on the physical field with the curvilinear coordinates (ξ,η) as the dependent variables and the cartesian coordinates (x,y) as the independent variables, with boundary conditions specified on a curved boundary Γ.



$\xi((x,y) \in \Gamma_2) = 0$
$\xi((x,y) \in \Gamma_4) = m = \text{const}$

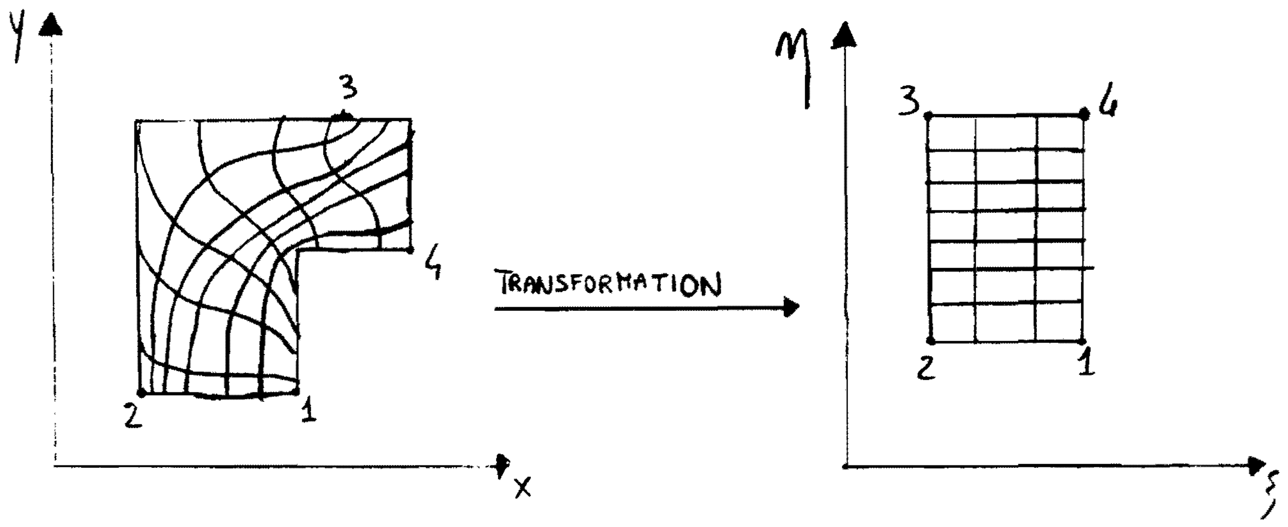
$\eta((x,y) \in \Gamma_1) = 0$
$\eta((x,y) \in \Gamma_3) = n = \text{const}$

where $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4$.

Although linear, the problem has implicitly defined boundary conditions, which make it very hard to handle.

The problem may be simplified for computation by first transforming it so that the physical cartesian coordinates (x,y) become the dependent variables, with the curvilinear coordinates (ξ,η) as the independent variables.
Then, at the expense of a nonlinear system, the boundary conditions are simplified to explicitly given values at a rectangle. The boundary value problem in the transformed field then generates the values of the physical cartesian coordinates, $x(\xi,\eta)$ and $y(\xi,\eta)$, in the transformed field. The boundary is now built up from segments of constant ξ or η, i.e. vertical or horizontal lines.

Example:



Our task is now to rewrite the equations (3.1) in terms of the new variables $x(\xi,\eta)$ and $y(\xi,\eta)$.
Let us consider a function $f = f(\xi(x,y), \eta(x,y))$. Then

$$\frac{\partial f}{\partial x} = f_\xi \xi_x + f_\eta \eta_x \ , \quad \frac{\partial f}{\partial y} = f_\xi \xi_y + f_\eta \eta_y .$$

If $f = x$ we have $\frac{\partial f}{\partial x} = 1$ and $\frac{\partial f}{\partial y} = 0.$
If $f = y$ we have $\frac{\partial f}{\partial x} = 0$ and $\frac{\partial f}{\partial y} = 1.$
So we get the following system:

$$\begin{bmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{bmatrix} \begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} .$$

Then it follows that

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

and we have the following relations:

$$\xi_x = \frac{y_n}{J} \quad , \quad \xi_y = -\frac{x_\eta}{J} ,$$
$$\eta_x = -\frac{y_\xi}{J} \quad , \quad \eta_y = \frac{x_\xi}{J} , \qquad (3.2)$$

where $J = x_\xi y_\eta - x_\eta y_\xi$.

Let us compute $\frac{\partial^2 f}{\partial x^2}$ and $\frac{\partial^2 f}{\partial y^2}$.

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}\left[\frac{\partial f}{\partial x}\right] = \frac{\partial}{\partial x}(f_\xi \xi_x - f_\eta \eta_x) =$$

$$= (f_{\xi\xi}\xi_x + f_{\xi\eta}\eta_x)\xi_x + f_\xi \xi_{xx} + (f_{\eta\xi}\xi_x + f_{\eta\eta}\eta_x)\eta_x + f_\eta \eta_{xx}.$$

If $f = x$ then $\frac{\partial^2 f}{\partial x^2} = 0$ i.e.

$$x_{\xi\xi}\xi_x^2 + 2x_{\xi\eta}\xi_x\eta_x + x_\xi \xi_{xx} + x_{\eta\eta}\eta_x^2 + x_\eta \eta_{xx} = 0. \qquad (3.3a)$$

Similarly,

$$\frac{\partial^2 f}{\partial y^2} = f_{\xi\xi}\xi_y^2 + f_{\xi\eta}\eta_y\xi_y + f_\xi \xi_{yy} + f_{\eta\xi}\xi_y\eta_y + f_{\eta\eta}\eta_y^2 + f_\eta \eta_{yy}.$$

If $f = x$ then $\frac{\partial^2 f}{\partial y^2} = 0$ i.e.

$$x_{\xi\xi}\xi_y^2 + 2x_{\xi\eta}\xi_y\eta_y + x_\xi \xi_{yy} + x_{\eta\eta}\eta_y^2 + x_\eta \eta_{yy} = 0. \qquad (3.3b)$$

Summation of (3.3a) and (3.3b) yields

$$x_{\xi\xi}(\xi_x^2 + \xi_y^2) + 2x_{\xi\eta}(\xi_x\eta_x + \xi_y\eta_y) + x_\xi(\xi_{xx} + \xi_{yy}) + x_{\eta\eta}(\eta_x^2 + \eta_y^2) + x_\eta(\eta_{xx} + \eta_{yy}) = 0.$$

Taking into account the defining system (3.1) and the relations (3.2) we get

$$x_{\xi\xi}\frac{1}{J^2}(x_\eta^2 + y_\eta^2) + 2x_{\xi\eta}\frac{1}{J^2}(-x_\eta x_\xi - y_\eta y_\xi) + x_{\eta\eta}\frac{1}{J^2}(x_\xi^2 + y_\xi^2) + x_\xi P + x_\eta Q = 0.$$

The same for $f = y$ after changing x into y.
Hence, after changing the dependent and independent variables with each other, the system (3.1) takes the following form:

$$\begin{cases} g^{11}x_{\xi\xi} + 2g^{12}x_{\xi\eta} + g^{22}x_{\eta\eta} + x_\xi P + x_\eta Q = 0 \\ g^{11}y_{\xi\xi} + 2g^{12}y_{\xi\eta} + g^{22}y_{\eta\eta} + y_\xi P + y_\eta Q = 0 \end{cases} \qquad (3.4)$$

where:

$$g^{11} = \frac{1}{J^2}(x_\eta^2 + y_\eta^2), \quad g^{22} = \frac{1}{J^2}(x_\xi^2 + y_\xi^2), \quad g^{12} = -\frac{1}{J^2}(x_\xi x_\eta + y_\xi y_\eta).$$

If we consider, as original generating system, the Laplace system ($P = Q = 0$), the system in the new coordinates becomes

$$\begin{cases} g^{11}x_{\xi\xi} + 2g^{12}x_{\xi\eta} + g^{22}x_{\eta\eta} = 0 \\ g^{11}y_{\xi\xi} + 2g^{12}y_{\xi\eta} + g^{22}y_{\eta\eta} = 0. \end{cases} \qquad (3.5)$$

The equations (3.4) and (3.5) are of the same type as the original ones, but are more complicated in that they are non linear and contain more terms. The domain, on the other hand, is greatly simplified since it is, regardless of its shape, transformed to a fixed rectangular region. This facilitates the imposition of boundary conditions and is the primary feature which makes grid generation such a valuable and important tool in the numerical solution of partial differential equations on arbitrary domains. At the same time, the rectangular domain allows us to use well-known finite difference algorithms for solving the equations for x and y.

4. ORTHOGONALITY

When the boundary values are fixed, one way to change the angle of intersection of ξ-lines and η-lines with the boundary is by a suitable choice of the control functions.

Such a technique enables us to constrain the transverse lines to be locally straight and orthogonal to the boundary (see [2]).

Let us suppose that the source terms have the form

$$P = \Phi(\xi,\eta)\ (\xi_x^2+\xi_y^2),$$

$$Q = \Psi(\xi,\eta)\ (\eta_x^2+\eta_y^2),$$

where the auxiliary functions Φ and Ψ are yet to be specified.

If we rewrite P and Q in terms of the cartesian coordinates, we get:

$$P = \Phi \frac{1}{J^2}(y_\eta^2+x_\eta^2) = \Phi \frac{1}{J^2} g^{11},$$

$$Q = \Phi \frac{1}{J^2}(x_\xi^2+y_\xi^2) = \Psi \frac{1}{J^2} g^{22}.$$

Upon introducing these terms, the equations (3.4) assume the form

$$\begin{cases} g^{11}(x_{\xi\xi}+\Phi x_\xi)+2g^{12}x_{\xi\eta}+g^{22}(x_{\eta\eta}+\Psi x_\eta)=0 \\ g^{11}(y_{\xi\xi}+\Phi y_\xi)+2g^{12}y_{\xi\eta}+g^{22}(y_{\eta\eta}+\Psi y_\eta)=0. \end{cases} \tag{4.1}$$

Given a set of boundary values (x,y) on the boundary of the computational domain, we can determine the functions Φ and Ψ by requiring that the given boundary values satisfy appropriate limiting forms of equations (4.1) along the boundary of the computational domain.

We can obtain the equations that define Φ and Ψ in terms of the boundary conditions by imposing the two conditions that the transverse coordinate curves be locally straight and orthogonal to the boundary.

Let ϕ and ψ denote the functions Φ and Ψ along the boundary.

To find one equation that determines the function ϕ along either of the horizontal boundaries $\eta=\eta_b=$ const, we first eliminate ψ between the two equations (4.1).

Hence the equation by which we will obtain ϕ is the following:

$$g^{11}\,[y_\eta(x_{\xi\xi}+\phi x_\xi)-x_\eta(y_{\xi\xi}+\phi y_\xi)] = -y_\eta^2\,[2g^{12}(x_\eta/y_\eta)_\xi+g^{22}(x_\eta/y_\eta)_\eta]. \tag{4.2}$$

If we consider the slope of the coordinate curves

$$\frac{dx}{dy} = \frac{x_\xi\, d\xi + x_\eta\, d\eta}{y_\xi\, d\xi + y_\eta\, d\eta},$$

then we can say that the ratio x_η/y_η is the slope of the family of coordinate curves $\xi=$ const that are transverse to the horizontal boundaries $\eta=\eta_b=$ const.

We impose now the constraint that these coordinate curves $\xi = \text{const}$ be locally straight (i.e. have zero curvature) in the neighbourhood of the boundary, which is:

$$(x_\eta / y_\eta)_\eta = 0 \quad \text{on} \quad \eta = \eta_b. \tag{4.3}$$

Let us impose the further constraint that the coordinate curves $\xi = \text{const}$ be locally orthogonal to the boundary $\eta = \eta_b$.
Let r_ξ denote the vector that is locally tangent to a coordinate curve $\eta = \text{const}$, i.e. $r_\xi = (x_\xi, y_\xi)$.
Similarly, the local tangent vector to a coordinate curve $\xi = \text{const}$ is $r_\eta = (x_\eta, y_\eta)$.
The two families of coordinate curves are then orthogonal if and only if

$$r_\xi \cdot r_\eta = 0$$

i.e.:

$$x_\xi x_\eta + y_\xi y_\eta = 0 \quad \text{on} \quad \eta = \eta_b. \tag{4.4}$$

When we evaluate the equation (4.2) at the boundary $\eta = \eta_b$, under the conditions (4.3) and (4.4), we get

$$g^{11} [y_\eta (x_{\xi\xi} + \phi x_\xi) - x_\eta (y_{\xi\xi} + \phi y_\xi)] = 0 \quad \text{on} \quad \eta = \eta_b. \tag{4.5}$$

Now we solve this equation for the parameter ϕ.
From (4.5) we have

$$x_{\xi\xi} + \phi x_\xi - \frac{x_\eta}{y_\eta} (y_{\xi\xi} + \phi y_\xi) = 0. \tag{4.6}$$

From (4.4):

$$x_\eta / y_\eta = - y_\xi / x_\xi \, ;$$

hence the equation (4.6) takes the form

$$x_{\xi\xi} + \phi x_\xi + \frac{y_\xi}{x_\xi} y_{\xi\xi} + \frac{y_\xi^2}{x_\xi} \phi = 0 \quad \Rightarrow$$

$$\phi = - \frac{(x_\xi x_{\xi\xi} + y_\xi y_{\xi\xi})}{(x_\xi^2 + y_\xi^2)}. \tag{4.7}$$

The latter is an universally valid equation that can be used to compute the numerical value of ϕ at each grid point on the boundary in terms of the boundary values of x and y.
In fact, numerically, the first and the second derivatives of x and y are computed in terms of the values of x and y.

The corresponding expression that determines ψ along the vertical boundaries $\xi = \xi_b$ can be obtained from equation (4.7) by putting ψ and η in place of ϕ and ξ, i.e.:

$$\psi = -\frac{(x_\eta x_{\eta\eta} + y_\eta y_{\eta\eta})}{(x_\eta^2 + y_\eta^2)}. \tag{4.8}$$

Once ψ is defined at each mesh point of the vertical boundaries approximations of the corresponding values of Ψ at interior mesh points can be computed by linear interpolation along horizontal mesh lines.

Similarly, the internal values of Φ are computed by interpolation along the vertical mesh lines.

Equations (4.1) can then be solved and they will generate the grid in the physical domain.

The procedure for evaluating the functions Φ and Ψ in the source terms insures that the transverse grid lines will be locally near-orthogonal to the boundaries.

5. NUMERICAL IMPLEMENTATION

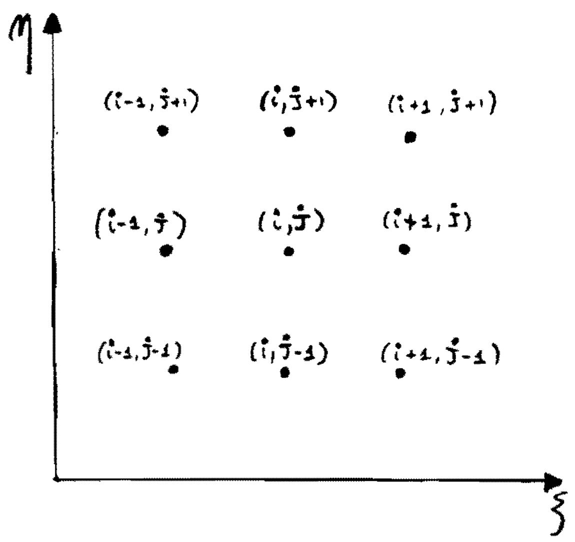
As said in section 3, for the transformed equations (3.5), the domain is always a fixed rectangular region and, since the increments of the curvilinear coordinates are arbitrary, the computation can always be done on a fixed uniform square grid. Spatial derivatives can therefore be represented by conventional finite-difference expressions.

Implementation of an elliptic generation system then is accomplished by devising an algorithm for the numerical solution of the partial differential equations comprising the generation system. The usual approach is to replace all derivatives in the partial differential equations by second order central difference expressions and then to solve the resulting system of algebraic difference equations by iteration.

5.1. Discrete representation of derivatives

Approximate values of the spatial derivatives of the functions which appear in the transformed equations (3.5) may be found at a given point in terms of the values of the functions at that point and at neighbouring points.

As noted earlier, with the problem in the transformed space, only uniform square grids need be considered, hence the standard forms for difference representation of derivatives may be used. For example, in two dimensions, the first, second, and mixed partial derivatives with respect to the curvilinear coordinates ξ and η are ordinarily represented at an interior point (i,j) by finite difference expressions which contain function values at no more than the nine points shown below.



Examples of finite difference approximations of this type are:

$$(x_\xi)_{i,j} = \frac{1}{2}(x_{i+1,j} - x_{i-1,j}) \tag{5.1a}$$

$$(x_\eta)_{i,j} = \frac{1}{2}(x_{i,j+1} - x_{i,j-1}) \tag{5.1b}$$

$$(x_{\xi\xi})_{i,j} = x_{i+1,j} - 2x_{i,j} + x_{i-1,j} \tag{5.1c}$$

$$(x_{\eta\eta})_{i,j} = x_{i,j+1} - 2x_{i,j} + x_{i,j-1} \tag{5.1d}$$

$$(x_{\xi\eta})_{i,j} = \frac{1}{4}(x_{i+1,j+1} - x_{i+1,j-1} - x_{i-1,j+1} + x_{i-1,j-1}). \tag{5.1e}$$

Analogous formulas can also be written for the function $y(\xi,\eta)$ which appears in the equations (3.5).

The centered difference representations (5.1) are second order accurate.

5.2. Convergence of finite difference scheme

As we can see from the equations (3.4) and (3.5), most generation systems are not linear, so that the difference equations are non linear and convergence depends therefore on the initial guess in iterative solutions. The algebraic grid generation procedures can serve, for instance, to generate this initial guess.

Since the coordinate lines tend to concentrate near a concave boundary, very sharp concave corners may cause problems with the convergence of the generation equations.

Moreover, the finite difference equations involve only a few mesh values, while the total number of mesh values can be very large so that the "geometric information" diffuses slowly. Therefore there are points which are very slow to move from the initial guess. Convergence in such a case can be very slow.

We have handled these problems by first converging the solution with a course grid and, after convergence, the number of grid points has been increased and so on until the required grid was reached. In order to apply such a multigrid method, the numbers of meshes in x and y direction must be powers of two, allowing an iteration based on halving the meshsize.

In the appendix a listing is presented of a Turbo Pascal 5.0 implementation.

6. EXAMPLES

In the picture 1 a grid generated by solving the Laplace system is shown.

In the picture 2 we can see the effect of the source terms in the Poisson system (the points where the sources are applied are marked by small circles).

The third picture is obtained by solving the Poisson system and considering a doubled mesh-size.

In the pictures 4, 5, 6 the same operations are applied to another region.

The picture 7 represents an attempt to apply the procedure for the orthogonality presented in section 4.
It is immediate to realize that, for boundaries which have only corners and no smooth curvatures, the orthogonality cannot be reached in the way previously suggested. The reason for that is that, since the expressions of the control functions involve the curvature of the boundary which is concentrated here at the corners, the source terms are zero everywhere apart from lines enumerating from these corners.
This is, of course, a degenerate situation.

7. CONCLUSIONS

To summarize, the necessary basic information, from both the standpoint of mathematical background and from that of coding implementation, for an analytical method of constructing grids on general regions, has bee provided.

A method for controlling grids generated by an elliptic system has also been presented and implemented. Such a control is done by introducing source terms whose mathematical form is independent of the boundary shape.

Apart from the density of the coordinate lines in general, such a procedure allows one to control directly the angle of intersection between the two families of coordinate curves at boundaries.
In particular, we have imposed the constraint that the two families be locally orthogonal at the boundaries. The source terms are written with free parameters that are evaluated locally at the boundaries using limiting forms of the generating elliptic equations; then these parameters are interpolated into the interior of the computational domain.

Finite difference representations of curvilinear coordinate systems have been constructed by first transforming derivatives with respect to the cartesian coordinates into expressions involving derivatives with respect to the curvilinear coordinates. The derivatives with respect to the curvilinear coordinates are then replaced by finite difference expressions on the uniform grid in the transformed region. A fast convergence is obtained by implementing it as a multigrid scheme.

The technique that we have presented for the generation of curvilinear coordinate systems and then of computational meshes, is characterized by both simplicity and power. In fact all computations are done on a fixed square grid without any restriction to certain boundary shapes.

REFERENCES

[1] Eiseman, Peter R., "Automatic Algebraic Coordinate Generation", Numerical Grid Generation, Ed. Joe F. Thompson, North Holland, **447** (1982).

[2] J.F. Middlecoff and P.D. Thomas, "Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations", AIAA Journal, 18, **652** (1980).

[3] Joe F. Thompson, Z.U.A. Warsi, C. Wayne Mastin, "Numerical Grid Generation", North Holland.

APPENDIX

```
program meshgen;

    {positive source pushes the (curvilinear) coordinate to higher values,
                                                              negative to lower}

uses
    graph, crt;
label
    einde;
const
  geom    = 1;        { geom = 1/2/3 are some test geometries}
  ni      = 8;
  nj      = 32;
  relax   = 0.0;      { relaxation parameter 0.0 <= .. <= 1.0 }
  eps     = 0.0001;    { error level}
  countmax = 15;
  source  = true ;     { = True / False ; toggles source on/off }
  ortho   = false;     { = True / False ; toggles "orthogonal source" on/off}
type
   arr = array[0..ni,0..nj] of real;
var
  x,y,p1,p2,phi,psi        :arr;
  i,j,nsub,count           :integer;
  xmax,xmin,ymax,ymin      :real;
  diff,x0,y0               :real;

procedure wait;
var
 answ  : char;
begin
 repeat until keypressed;
 answ:=readkey;
end;
```

```
function chckparm:boolean;
var
  ierr :integer;
begin
  ierr:=0;
  if ni<> round(exp(round(ln(ni)/ln(2))*ln(2))) then ierr:=1;
  if nj<> round(exp(round(ln(nj)/ln(2))*ln(2))) then ierr:=2;
  if (geom<>1) and (geom<>2) and (geom<>3) then ierr:=3;
  if (relax<0) or (relax>1) then ierr:=4;
  if countmax<1 then ierr:=5;
  if ierr<>0 then
  begin
    chckparm := False;
    case ierr of
     1: writeln('ni must be power of 2');
     2: writeln('nj must be power of 2');
     3: writeln('geom must be 1,2, or 3');
     4: writeln('relax must be between 0 and 1');
     5: writeln('countmax must be > 0');
    end;
    wait;
  end
  else chckparm:=True;
end;

function min(i,j:integer):integer;
begin
 if i<j then min:=i else min:= j;
end;

procedure initsource;
var
 i,j    :integer;
begin
 for i:=1 to ni-1 do
 begin
  for j:=1 to nj-1 do
  begin
   phi[i,j]:= 0.0;
   psi[i,j]:= 0.0;
   p1[i,j] := 0.0;
   p2[i,j] := 0.0;
  end;
 end;
end;
```

```
procedure boundary (m:integer);
var
   ni1,ni2,i,j                  :integer;
   nj1,nj2,nj3,nj4,nj5,nj6,nj7 :integer;
begin
 case m of
   1: begin
        xmax := 2.5;
        xmin :=-0.5;
        ymax := 2.5;
        ymin :=-0.5;
        ni1  := ni div 2;
        nj1  := nj div 2;
        ni2  := ni-ni1;
        nj2  := nj-nj1;
        for i:=0 to ni do
        begin
          x[i,0] := i/ni;
          y[i,0] := 0.0;
        end;
        for i:=0 to ni1 do
        begin
          x[i,nj] := 1+i/ni1;
          y[i,nj] := 2.0;
        end;
        for i:=ni1+1 to ni do
        begin
          x[i,nj] := 2.0;
          y[i,nj] := 2-(i-ni1)/ni2;
        end;
        for j:=1 to nj1 do
        begin
          x[0,j]  := 0.0;
          y[0,j]  := 2*j/nj1;
        end;
        for j:=nj1+1 to nj-1 do
        begin
          x[0,j]  := (j-nj1)/nj2;
          y[0,j]  := 2.0;
        end;
        for j:=1 to nj1 do
        begin
          x[ni,j] := 1.0;
          y[ni,j] := j/nj1;
        end;
        for j:=nj1+1 to nj-1 do
        begin
          x[ni,j] := 1+(j-nj1)/nj2;
          y[ni,j] := 1.0;
        end;
        if source then
        begin
          for j:=nj1-3 to nj1+3 do
          begin
            p1[1,j] := -20.0;          {pushes xi to lower "i"};
            p1[2,j] := -10.0;
            p1[ni-1,j] := -20.0;
            p1[ni-2,j] := -10.0;
          end;
        for i:=ni1-2 to ni1+2 do
        begin
          p2[i,nj-1] := 20.0;        {pushes eta to higher "j"};
          p2[i,nj-2] := 10.0;
        end;
      end;
    end;
2: begin
      xmax := 3.5;
      xmin :=-0.5;
      ymax := 3.5;
      ymin :=-0.5;
      nj1 := round(nj/10);
      nj2 := round(nj*2/10);
      nj3 := round(nj*3/10);
      nj4 := round(nj*4/10);
      nj5 := round(nj*5/10);
      nj6 := round(nj*6/10);
      nj7 := round(nj*7/10);
      for i:=0 to ni do
      begin
        x[i, 0] := i/ni;
        y[i, 0] := 0;
      end;
      for i:=0 to ni do
      begin
        x[i,nj] := 0;
        y[i,nj] := 2+i/ni;
      end;
      for j:=1 to nj2 do
      begin
        x[0,j] := 0;
        y[0,j] := 2*j/nj2;
      end;
      for j:=nj2+1 to nj6-2 do
      begin
        x[0,j] := 2*(j-nj2)/(nj6-nj2);
        y[0,j] := 2;
      end;
      x[0,nj6-1]:=1.9;
      y[0,nj6-1]:=1.9;
      x[0,nj6]:=2;
      y[0,nj6]:=2;
      x[0,nj6+1]:=1.9;
      y[0,nj6+1]:=2.1;
      for j:=nj6+2 to nj-1 do
      begin
        x[0,j] := 2-2*(j-nj6)/(nj-nj6);
        y[0,j] := 2;
      end;
      for j:=1 to nj3 do
      begin
        x[ni,j] := 1;
        y[ni,j] := j/nj3;
      end;
      for j:=nj3+1 to nj5 do
      begin
        x[ni,j] := 1+2*(j-nj3)/(nj5-nj3);
        y[ni,j] := 1;
      end;
    for j:=nj5+1 to nj7 do
    begin
     x[ni,j] := 3;
     y[ni,j] := 1+2*(j-nj5)/(nj7-nj5);
    end;
    for j:=nj7+1 to nj-1 do
    begin
     x[ni,j] := 3-3*(j-nj7)/(nj-nj7);
     y[ni,j] := 3;
    end;
    if source then
    begin
     for j:=nj3-1 to nj3+1 do
     begin
      p1[ni-1,j] := -20;
      p1[ni-2,j] := -10;
      p2[ni-1,j] := -20;
      p2[ni-2,j] := -10;
     end;
     for j:=nj2-1 to nj2+1 do
     begin
      p1[1,j] := -20;
      p1[2,j] := -10;
      p2[1,j] := -20;
      p2[2,j] := -10;
     end;
     for j:=nj5-1 to nj5+2 do
     begin
      p1[ni-1,j] := 20;
      p1[ni-2,j] := 10;
      p2[ni-1,j] := 5;
      p2[ni-2,j] := 5;
     end;
     for j:=nj6-1 to nj6+1 do
     begin
      p1[1,j] := 40;
      p1[2,j] := 20;
      p1[3,j] := 10;
      p2[1,j] := 0;
      p2[2,j] := 0;
     end;
     for i:=nj7-2 to nj7+1 do
     begin
      p1[ni-1,j] := 20;
      p1[ni-2,j] := 10;
      p2[ni-1,j] := -5;
      p2[ni-2,j] := -5;
     end;
    end;
   end;
3: begin
    xmax := 2.5;
    xmin :=-0.5;
    ymax := 2.5;
    ymin :=-0.5;
    nj1 := round(nj/3);
    nj2 := round(nj*2/3);
    for i:=0 to ni do
    begin
     x[i,0] := i/ni;
```

```
          y[i,0] := 0;
         end;
         for i:=0 to ni do
         begin
          x[i,nj] := 1+i/ni;
          y[i,nj] := 2;
         end;
         for j:=1 to nj2 do
         begin
          x[0,j] := 0;
          y[0,j] := 2*j/nj2;
         end;
         for j:=nj2+1 to nj-1 do
         begin
          x[0,j] := (j-nj2)/(nj-nj2);
          y[0,j] := 2;
         end;
         for j:=1 to nj1 do
         begin
          x[ni,j] := 1;
          y[ni,j] := j/nj1;
         end;
         for j:=nj1+1 to nj2 do
         begin
          x[ni,j] := 1+(j-nj1)/(nj2-nj1);
          y[ni,j] := 1;
         end;
         for j:=nj2+1 to nj-1 do
         begin
          x[ni,j] := 2;
          y[ni,j] := 1+(j-nj2)/(nj-nj2);
         end;
         if source then
         begin
          for j:=nj2-3 to nj2+3 do
          begin
           p1[1,j] := -20;
           p1[2,j] := -10;
          end;
          for j:=nj1-3 to nj1+3 do
          begin
           p1[ni-1,j] := 20;
           p1[ni-2,j] := 10;
          end;
         end;
        end;
     end;
   end;
```

```
procedure xynew (i,j,k:integer;r:real);
var
 xdi,xdj,ydi,ydj,xdij,ydij,jac2,g11,g12,g22,g,xnw,ynw :real;
begin
 xdi   := (x[i+k,j]-x[i-k,j])/2;
 xdj   := (x[i,j+k]-x[i,j-k])/2;
 ydi   := (y[i+k,j]-y[i-k,j])/2;
 ydj   := (y[i,j+k]-y[i,j-k])/2;
 xdij  := (x[i+k,j+k]-x[i-k,j+k]-x[i+k,j-k]+x[i-k,j-k])/4;
 ydij  := (y[i+k,j+k]-y[i-k,j+k]-y[i+k,j-k]+y[i-k,j-k])/4;
 jac2  := sqr(xdi*ydj-xdj*ydi);
 g11   := ydj*ydj+xdj*xdj;
 g12   :=-ydj*ydi-xdj*xdi;
 g22   := ydi*ydi+xdi*xdi;
 g    := 2*(g11+g22);
 xnw   := (g11*(x[i+k,j]+x[i-k,j]+phi[i,j]*xdi)+
              g22*(x[i,j+k]+x[i,j-k]+psi[i,j]*xdj)+
              2*g12*xdij+jac2*(p1[i,j]*xdi+p2[i,j]*xdj))/g;
 ynw   := (g11*(y[i+k,j]+y[i-k,j]+phi[i,j]*ydi)+
              g22*(y[i,j+k]+y[i,j-k]+psi[i,j]*ydj)+
              2*g12*ydij+jac2*(p1[i,j]*ydi+p2[i,j]*ydj))/g;
 x[i,j] := (1-r)*xnw+r*xnw;
 y[i,j] := (1-r)*ynw+r*ynw;
end;
```

```
procedure orthog;
var
 i,j                          :integer;
 xdi,xdii,xdj,xdjj,ydi,ydii,ydj,ydjj :real;
begin
 j:=0;
 while j<=nj do
 begin
  for i:=1 to ni-1 do
  begin
   xdi    := (x[i+1,j]-x[i-1,j])/2;
   xdii   := x[i+1,j]-2*x[i,j]+x[i-1,j];
   ydi    := (y[i+1,j]-y[i-1,j])/2;
   ydii   := y[i+1,j]-2*y[i,j]+y[i-1,j];
   phi[i,j] :=-(xdi*xdii+ydi*ydii)/(xdi*xdi+ydi*ydi);
  end;
  inc(j,nj);
 end;
 i:=0;
 while i<=ni do
 begin
  for j:=1 to nj-1 do
  begin
   xdj    := (x[i,j+1]-x[i,j-1])/2;
   xdjj   := x[i,j+1]-2*x[i,j]+x[i,j-1];
   ydj    := (y[i,j+1]-y[i,j-1])/2;
   ydjj   := y[i,j+1]-2*y[i,j]+y[i,j-1];
   psi[i,j] :=-(xdj*xdjj+ydj*ydjj)/(xdj*xdj+ydj*ydj);
  end;
  inc(i,ni);
 end;
 for j:=1 to nj-1 do
 begin
  for i:=1 to ni-1 do
  begin
   psi[i,j] := psi[0,j] + i*(psi[ni,j]-psi[0,j])/ni;
  end;
 end;
 for i:=1 to ni-1 do
 begin
  for j:=1 to nj-1 do
  begin
   phi[i,j] := phi[i,0] + j*(phi[i,nj]-phi[i,0])/nj;
  end;
 end;
end;
```

```
procedure interpol (k:integer);
var
 i,j,k2        :integer;
begin
k2 := k div 2;
i  := k;
 while i<=ni-k do
 begin
  j := k2;
  while j<=nj-k2 do
  begin
   x[i,j] := (x[i,j-k2]+x[i,j+k2])/2;
   y[i,j] := (y[i,j-k2]+y[i,j+k2])/2;
   inc(j,k);
  end;
  inc(i,k);
 end;
 j:=k2;
 while j<=nj-k2 do
 begin
  i:=k2;
  while i<=ni-k2 do
  begin
   x[i,j] := (x[i-k2,j]+x[i+k2,j])/2;
   y[i,j] := (y[i-k2,j]+y[i+k2,j])/2;
   inc(i,k);
  end;
 inc(j,k2);
 end;
end;


function fystoscr(i:integer;c:real):integer;
begin
   if i=1 then fystoscr := round(getmaxx*(c-xmin)/(xmax-xmin));
   if i=2 then fystoscr := round(getmaxy*(ymax-c)/(ymax-ymin));
end;

procedure gopoint(xc,yc:real);
begin
  moveto(fystoscr(1,xc),fystoscr(2,yc));
end;

procedure drawline (xc,yc:real);
begin
  lineto(fystoscr(1,xc),fystoscr(2,yc));
end;

procedure startgraf;
var
  graphdriver,graphmode    :integer;
begin
 graphdriver:=detect;{att400;}
 {graphmode:=att400hi;}
 initgraph(graphdriver,graphmode,'c:\turbo5\');
end;
```

```
procedure picture (m,k:integer);
var
  i,j   :integer;
begin
 clearviewport;
 if m=0 then
 begin
  gopoint (x[0,0],y[0,0]);
  for i:=1 to ni do
  begin
   drawline (x[i,0],y[i,0]);
  end;
  for j:=1 to nj do
  begin
   drawline (x[ni,j],y[ni,j]);
  end;
  for i:=ni-1 downto 0 do
  begin
   drawline (x[i,nj],y[i,nj]);
  end;
  for j:=nj-1 downto 0 do
  begin
   drawline (x[0,j],y[0,j]);
  end;
 end;
 if m=1 then
 begin
  i:=0;
  while i<=ni do
  begin
   gopoint(x[i,0],y[i,0]);
   j:=0;
   while j<=nj do
   begin
    drawline (x[i,j],y[i,j]);
    inc(j,k);
   end;
   inc(i,k);
  end;
  j:=0;
  while j<=nj do
  begin
   gopoint (x[0,j],y[0,j]);
   i:=0;
   while i<=ni do
   begin
    drawline (x[i,j],y[i,j]);
    inc(i,k);
   end;
   inc(j,k);
  end;
 end;
end;
```

```
begin                          {main program}
 if chckparm then
 begin
  initsource;
  startgraf;
  nsub := min(ni,nj);
  boundary(geom);
  if ortho then orthog;
  picture (0,1);                 { picture of geometry }
  wait;
  repeat
   interpol(nsub);                { interpolation for new points}
   nsub := nsub div 2;
   count := 0;
   repeat
    count := count+1;
    diff := 0;
    picture (1,nsub);
    i := nsub;
    while i<=ni-nsub do
    begin
     j := nsub;
     while j<=nj-nsub do
     begin
      x0 := x[i,j];
      y0 := y[i,j];
      xynew(i,j,nsub,relax);              { central algorithm }
      diff := diff+sqr(x[i,j]-x0)+sqr(y[i,j]-y0);
      if keypressed then begin wait;goto einde;end; {user's interruption}
      inc(j,nsub);
     end;
     inc(i,nsub);
    end;
    diff := sqrt(diff)*nsub/(ni+nj)/(xmax-xmin+ymax-ymin);  { scaled error}
   until (diff < eps) or (count > countmax);
  until nsub=1;
  einde:
  wait;
  closegraph;
 end;
end.
```

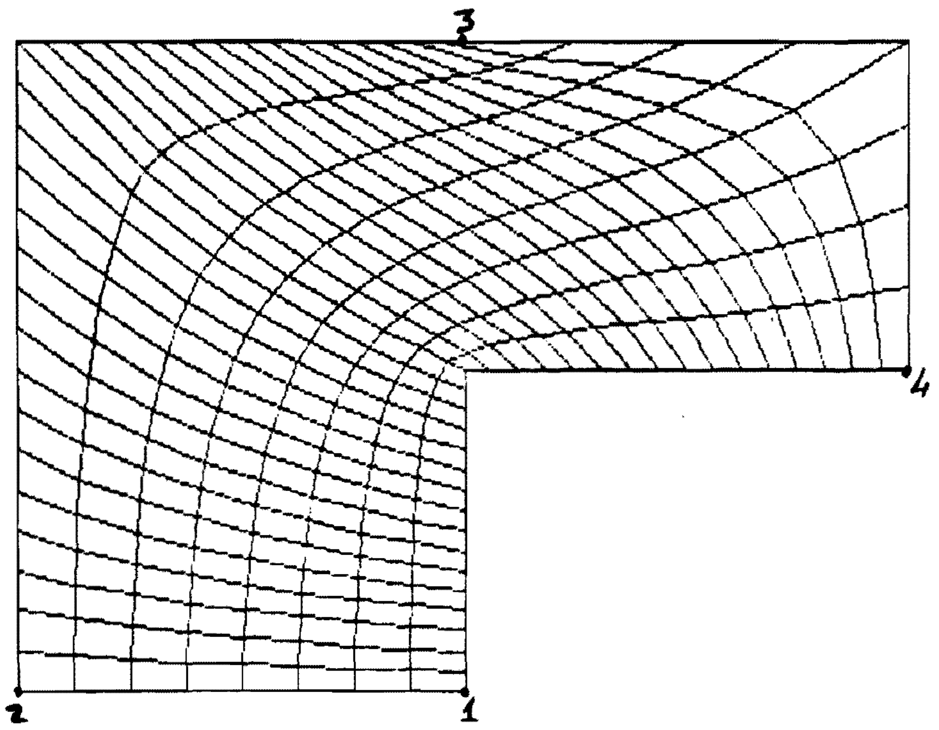


FIG. 1

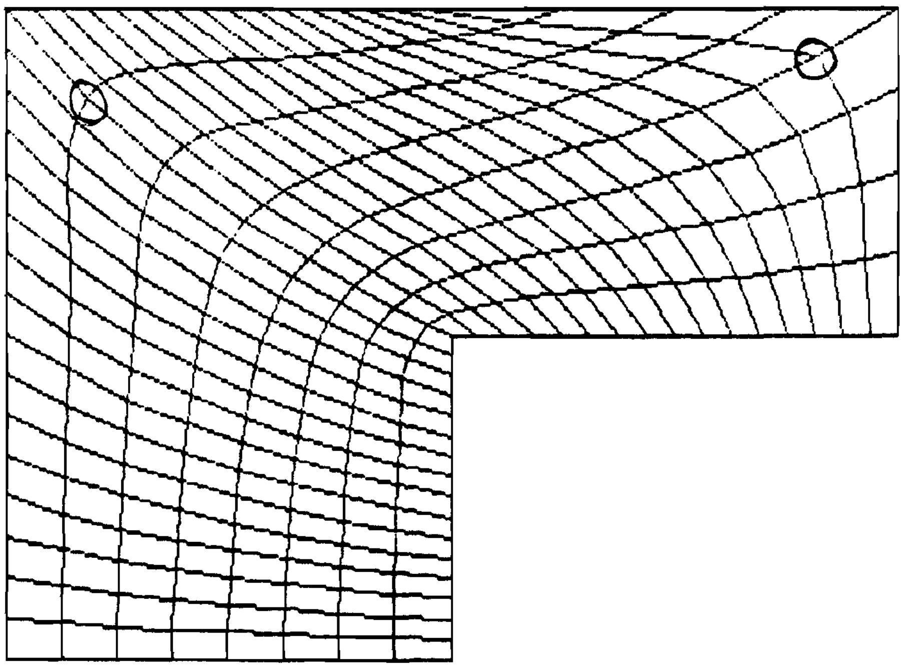


FIG. 2

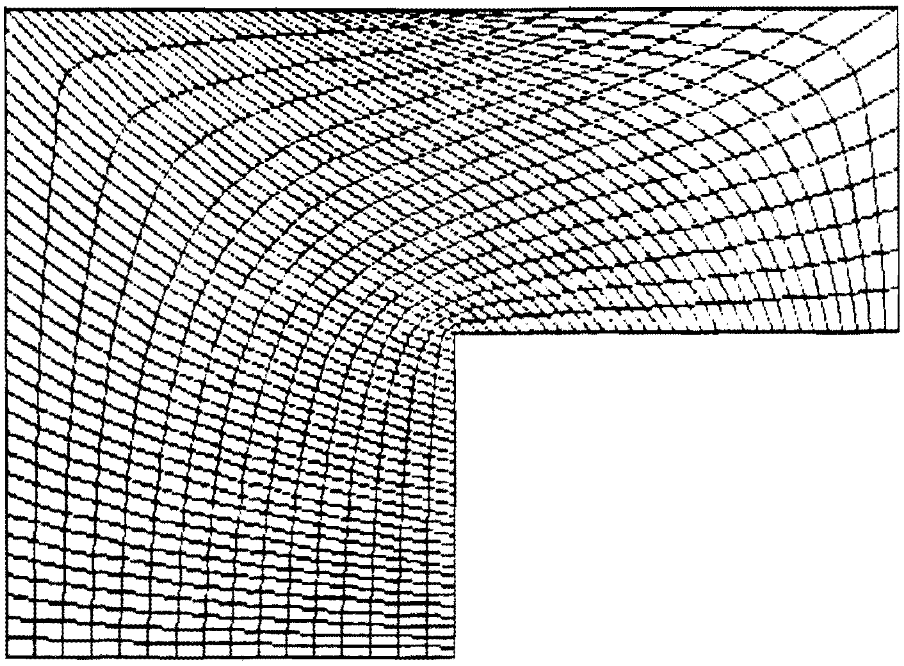


FIG. 3

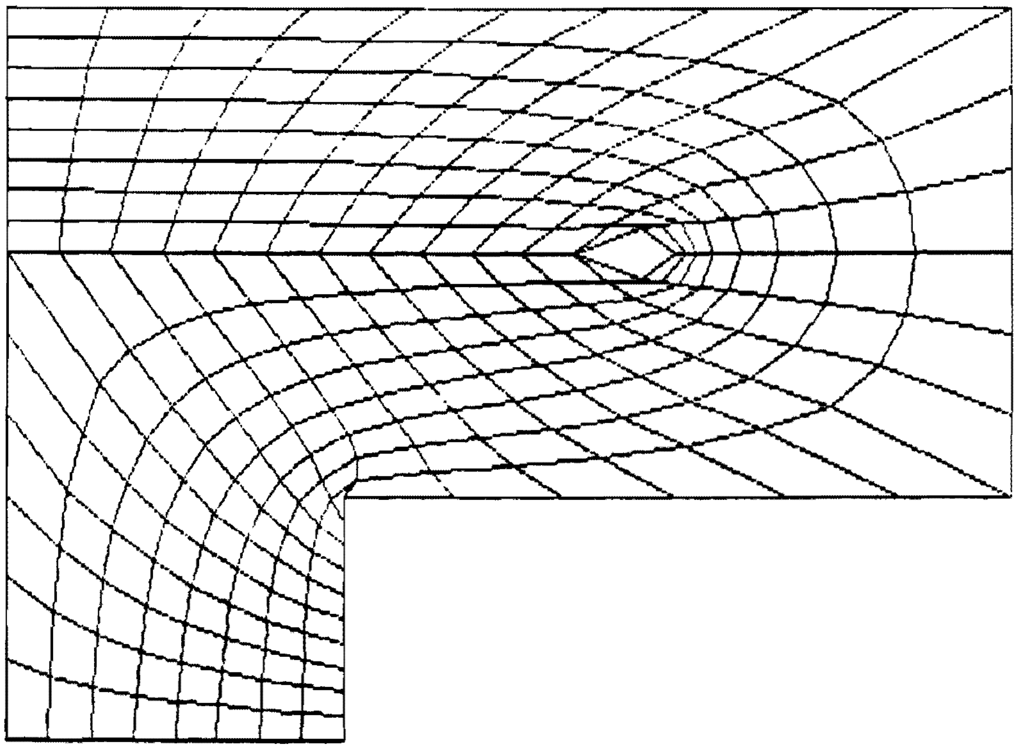


FIG. 4

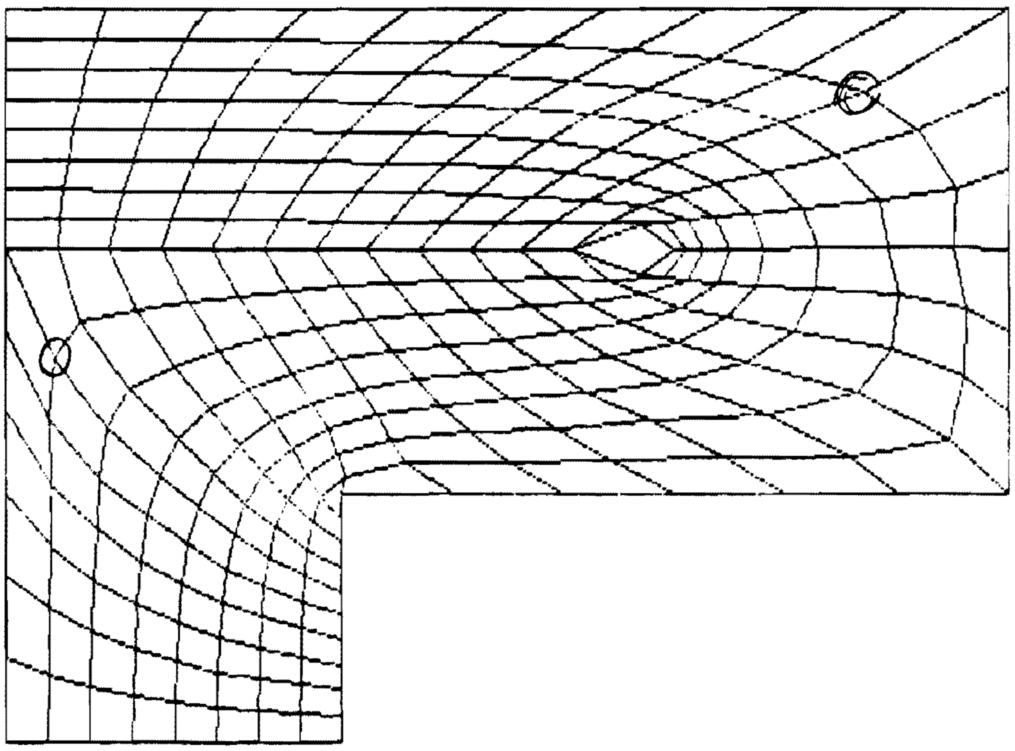


FIG. 5

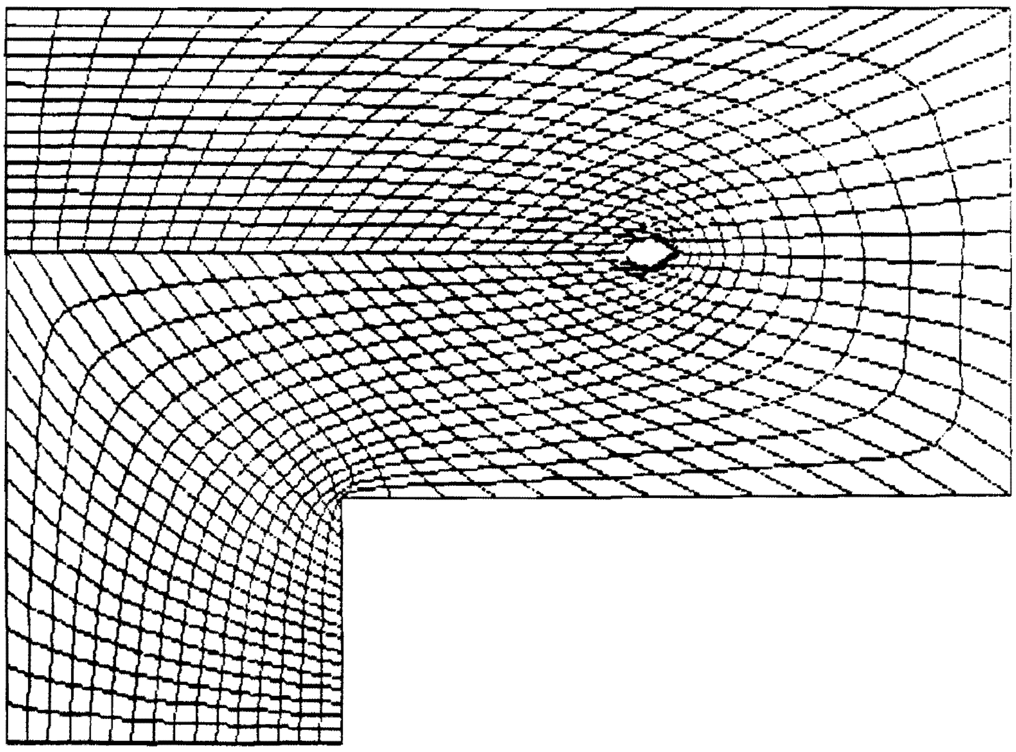


FIG. 6

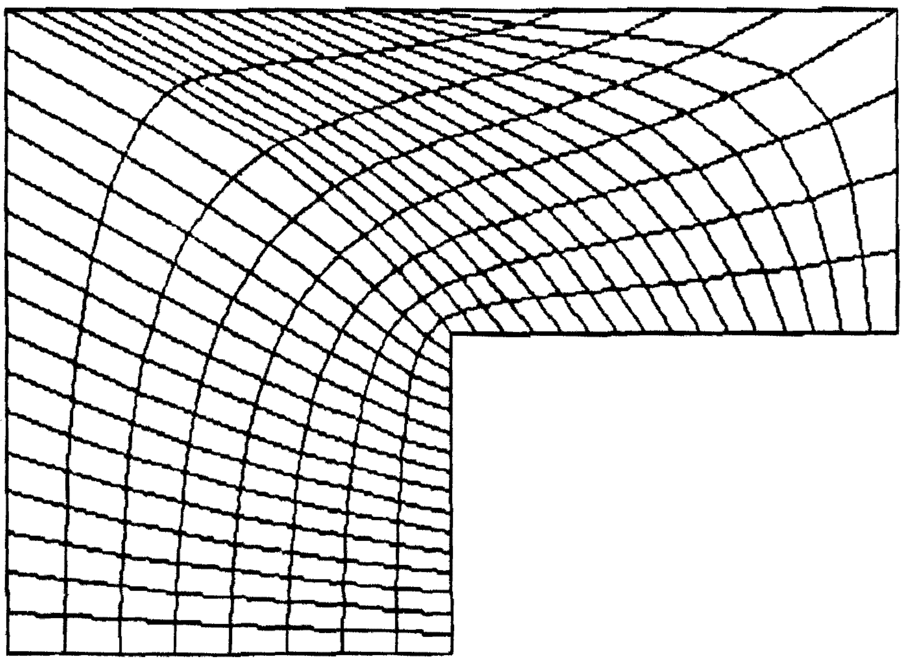


FIG. 7