COMPOSITE GRID CONSTRUCTION AND APPLICATIONS

Thesis by

Geoffrey Samuel Chesshire

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1986

(Submitted May 15, 1986)

Acknowledgements

I would like to thank my advisor, Heinz Kreiss, for his many helpful suggestions and for his infinite patience. A large part of the credit belongs to Bill Henshaw, whose collaboration made this work possible. I was able to complete this thesis only with the steady encouragement and support of my family and the special friends I have met while at Caltech. I would also like to thank Gerald Browning, who arranged for me to work several summers at the National Center for Atmospheric Research.

My work at Caltech has been supported by the Natural Science and Engineering Research Council of Canada through scholarships, the Department of Energy under contract DE-AA03-76SF0076, the Office of Naval Research under contract N00014-83-K-0422, and the National Science Foundation under contract ATM8201207. The computations were done on the Fluid Dynamics VAX 11/750 at Caltech. The figures were prepared using the NCAR graphics package and the manuscript was typeset using TEX.

Abstract

For many problems that involve the discrete representation of a function on a two-dimensional region, a single rectangular grid is inappropriate. However, any compact region with a smooth boundary can be discretised using a composite grid, which provides boundary-fitted coordinates and allows stretched coordinates for boundary layers and internal layers wherever these are needed. We have developed interactive software for specifying composite grids, and general-purpose subroutines for using them in discretising PDE boundary-value problems. The discretisation of an elliptic BVP with an internal layer and of a supersonic flow problem are given as examples of the uses of composite grids and the capabilities of the software.

Table of Contents

List of Figures

List of Tables

Introduction

For many problems that involve the discrete representation of a function on a two-dimensional region, a single rectangular grid is inappropriate. For example, a rectangular grid is poorly suited to the discretisation of a partial differential boundary-value problem on any region but a rectangle, as the boundary of the region will not in general correspond to a line of the grid. As a result, the accuracy of the solution may be significantly degraded, as demonstrated by Henshaw [12]. Many methods have been devised to construct a one-to-one transformation from a region bounded by piecewise smooth curves onto a rectangle. See, for example, the review article by Eiseman [9] and the conference proceedings edited by Thompson [22]. A cartesian coordinate system on the rectangle then corresponds to a curvilinear coordinate system on the region, and at least for the case of a simply-connected region, the boundary of the rectangle corresponds to the boundary of the region. A rectangular grid on the rectangle corresponds to a curvilinear grid on the region, and the boundary of the region corresponds everywhere to a gridline. One difficulty faced by all of these methods is that for a region of complex geometry they have very limited control over the distribution of coordinate lines, and hence of the gridlines. This difficulty is addressed with some success by the elliptic grid generation methods of Thompson and Mastin [15] and the adaptive grid generation methods of Brackbill and Saltzman [4]. However, even for a region of relatively simple geometry such as a simply-connected region consisting of two discs connected by a narrow strip, the density of gridlines will inevitably be greater on the narrow strip than on the discs. Another difficulty is that any transformation from a multiply-connected

region (other than a doubly-connected region) to a rectangle must have a singularity somewhere on the boundary. Both of these difficulties are consequences of using a single grid to cover the entire region.

An alternative is to cover the region with a composite grid consisting of several overlapping curvilinear grids, each of which is the image, under a smooth transformation, of a rectangular grid, and chosen so that the boundary of the region is composed entirely of edges of the curvilinear grids. The density of gridlines on each curvilinear grid is independent of the other grids and is determined by the transformation specified for that grid. Using enough of these transformations and grids it is possible to discretise problems on complicated regions for which a single grid is hopelessly inadequate. A composite grid can be constructed for any compact region with a piecewise smooth boundary, even for a multiply-connected region. It provides boundary-fitted coordinates and allows stretched coordinates for boundary layers and internal layers wherever these are needed. The concept of a composite grid extends easily to regions with cylindrical or toroidal symmetry and to n-dimensional regions.

We have developed software for the interactive construction of composite grids for compact regions in the plane and for regions with cylindrical or toroidal symmetry, and general-purpose subroutines for using them in discretising PDE boundary-value problems. These include subroutines for reading in a data file containing a description of a composite grid, for calculating the coefficients in the centered second-order discretisation of a linear combination of derivatives on a nine-point stencil, for interpolating data between the grids of a composite grid in the regions of their overlap, and for the second-order discretisation and direct solution of a second-order linear elliptic equation with mixed boundary conditions on a composite grid. We have generalised the procedure of B. Kreiss [14] for construction of a pair of overlapping grids for a simply-connected region with a smooth boundary

so that we can construct a composite grid for any compact two-dimensional region with a piecewise-smooth boundary, even a multiply-connected region periodic in one or both directions. Further generalisations we have included allow for the use of stretched coordinates on the component grids to resolve boundary layers and internal layers, the discretisation of a PDE and its boundary conditions to any order of accuracy, the interpolation of function values between the component grids to any order of accuracy, and the construction of sequences of composite grids for use with multigrid methods. The discretisations of an elliptic boundary value problem with an internal layer and of a supersonic flow problem with multiple shocks are given as examples of the uses of composite grids and the capabilities of the software.

CHAPTER 1

Curvilinear Grids

Numerical simulations in fluid mechanics and many other problems involving the computation of a solution field on a bounded region in the plane are often carried out on a rectangular computational grid chosen so that the boundary of the region follows lines of the grid. This is only possible for regions bounded by curves consisting of line segments intersecting at right angles. Although regions bounded by such curves are involved in many interesting fluid flow problems, numerical simulations are carried out on such regions less often for this reason than because of the ease of accurately applying boundary conditions on a grid fitted to the boundary of the region. For some applications it may not be important to use a boundary-fitted grid; a rectangular grid may suffice. However, boundary conditions for the discretisation of partial differential equations are difficult to apply when the grid is not boundary-fitted, and for a problem with a boundary layer, truncation error in the boundary condition results in an inaccurate solution. Examples of this effect are given in [12]. Boundary conditions for a partial differential equation may be accurately discretised on a region with a piecewise-smooth boundary if a boundary-fitted curvilinear grid can be found. For this reason the construction of such grids is of interest.

1.1 Notation for Curvilinear Grids

We shall consider only curvilinear grids in two dimensions. Although the concepts discussed here extend easily to three or more dimensions, the notation becomes

difficult and regions and their boundaries become difficult to describe or imagine. Since the principal use for curvilinear grids is the discretisation of PDE boundary-value problems by finite-difference methods, we shall consider only "logically rectangular" grids, those on which the gridpoints can be labelled by two indices (i, j) so that gridlines drawn through all of the gridpoints with the same index i do not intersect, and likewise for gridlines of constant j. Such curvilinear grids may be continuously deformed from a uniform rectangular grid. This leads to the following definition of a curvilinear grid.

Definition 1.1.1 *A curvilinear grid for a compact region* $\mathbf{D}$ *in the plane, on a cylinder or on a torus is a continuous transformation* $\mathbf{d}$ *from the unit square onto* $\mathbf{D}$,

$$\mathbf{D} = \{\mathbf{d}(r, s): 0 \leq r \leq 1, 0 \leq s \leq 1\},$$

and a uniform rectangular grid $\mathbf{G}$ *on the unit square,*

$$\mathbf{G} = \left\{ \left(\frac{i-1}{m-1}, \frac{j-1}{n-1} \right) : i = 1, \ldots, m,\ j = 1, \ldots, n \right\}.$$

We shall henceforth say "grid" whenever it is clear from the context that we mean "curvilinear grid." Just how smooth the transformation $\mathbf{d}$ needs to be will depend on the specific application the grid is to be used for.

The boundary of a multiply-connected region, such as an annulus, consists of two or more disjoint curves. Under any continuous transformation $\mathbf{d}$ from the unit square onto such a region $\mathbf{D}$, the boundary curves of $\mathbf{D}$ are the image of parts of the boundary of the unit square; other parts of the boundary of the unit square are mapped onto curves called "branch cuts" in the interior of $\mathbf{D}$ that connect the boundary curves. Branch cuts may also be introduced, as explained below, so that the cylinder and the torus may be represented in the plane. However, these need not add to the confusion if they are chosen to follow the above branch cuts wherever they intersect $\mathbf{D}$.

A cylinder may be represented in the plane by a "period strip" bounded by a pair of parallel lines $\mathbf{C}_1$ and $\mathbf{C}_2$ which correspond to a single line, a branch cut, on the cylinder. If corresponding points on $\mathbf{C}_1$ and $\mathbf{C}_2$ are parametrised by $\mathbf{c}_1(\xi) = \mathbf{a}_1 + \mathbf{b}\xi$ and $\mathbf{c}_2(\xi) = \mathbf{a}_2 + \mathbf{b}\xi$, any function on the cylinder corresponds to a periodic function in the plane, with periodicity $\mathbf{c}_{12} = \mathbf{a}_1 - \mathbf{a}_2$. It is sometimes convenient to choose a curve other than a straight line for the branch cut on the cylinder. In that case the period strip in the plane is bounded by curves $\mathbf{C}_1$ and $\mathbf{C}_2$ on which corresponding points may be parametrised by functions $\mathbf{c}_1$ and $\mathbf{c}_2$ which differ by the constant vector $\mathbf{c}_{12} = \mathbf{c}_1(\xi) - \mathbf{c}_2(\xi)$. For example, if the the image of the boundary of the unit square under the transformation $\mathbf{d}$ crosses the branch cut on the cylinder, the corresponding transformation from the unit square onto the period strip in the plane is discontinuous. To avoid this problem the branch cut must be chosen so that the image of the boundary of the unit square under the transformation $\mathbf{d}$ does not cross it; this is achieved by choosing the branch cut in the cylinder so that it follows a branch cut due to the transformation $\mathbf{d}$ wherever it intersects $\mathbf{D}$. At the risk of some confusion, we use the same variables (x, y) to label coordinates on both the cylinder and the period strip in the plane, and use the same name for a function on the cylinder and the corresponding function on the period strip.

A torus may likewise be represented in the plane by a "period box" bounded by a trapezoid. Each pair of opposite sides of the trapezoid corresponds to a single line segment, a branch cut, on the torus. More generally, if the branch cuts on the torus are chosen to be curves, then the period box in the plane is bounded by two pairs of curves $(\mathbf{C}_1, \mathbf{C}_2)$ and $(\mathbf{C}_3, \mathbf{C}_4)$ on which corresponding points may be parametrised by two pairs of functions $(\mathbf{c}_1(\xi), \mathbf{c}_2(\xi))$ and $(\mathbf{c}_1(\xi), \mathbf{c}_2(\xi))$ which differ by the constant vectors $\mathbf{c}_{12} = \mathbf{c}_1(\xi) - \mathbf{c}_2(\xi)$ and $\mathbf{c}_{34} = \mathbf{c}_3(\xi) - \mathbf{c}_4(\xi)$. Any function on the torus corresponds to a doubly-periodic function in the plane with periods

$\mathbf{c}_{12}$ and $\mathbf{c}_{34}$. The branch cuts on the torus should be chosen so that the image of the boundary of the unit square under the transformation $\mathbf{d}$ does not cross them. We use the same variables (x, y) to label coordinates on both the torus and the period box in the plane, and use the same name for a function on the torus and the corresponding function on the period box.

1.2 PDEs on Curvilinear Grids

The principal use for curvilinear grids is in discretising PDE boundary-value problems using finite-difference methods on a region $\mathbf{D}$ whose boundary consists of smooth curves. To be useful for this purpose the grid must satisfy certain smoothness constraints that depend on the PDE. If the grid is constructed as the image under the transformation $\mathbf{d}$ of a rectangular grid, the constraints on the smoothness of the grid can be expressed as constraints on the transformation $\mathbf{d}$. Consider a PDE boundary value problem

$$\mathbf{F}\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \mathbf{u}\right) = 0 \quad \text{for} \quad (x, y) \in \mathbf{D} \tag{1.2.1}$$

where $\mathbf{F}$ represents the PDE in the interior of $\mathbf{D}$ and the boundary conditions on the boundary of $\mathbf{D}$, and it is understood that higher derivatives may be involved. We can define a function $\mathbf{u}'$ which represents $\mathbf{u}$ in cartesian coordinates on the unit square,

$$\mathbf{u}'(r, s) = \mathbf{u}(\mathbf{d}(r, s)) \quad \text{for} \quad (r, s) \in [0, 1]^2. \tag{1.2.2}$$

If the Jacobian of the transformation $\mathbf{d}$ is non-singular, we can use it to express the problem $\mathbf{F}$ in (r, s) coordinates. Let

$$J(r, s) = \frac{\partial \mathbf{d}}{\partial (r, s)} = \begin{pmatrix} x_r & x_s \\ y_r & y_s \end{pmatrix}. \tag{1.2.3}$$

Then the inverse of $J(r, s)$ is

$$J^{-1}(r, s) = \frac{1}{x_r y_s - y_r x_s} \begin{pmatrix} y_s & -x_s \\ -y_r & x_r \end{pmatrix} = \begin{pmatrix} r_x & r_y \\ s_x & s_y \end{pmatrix}. \tag{1.2.4}$$

With the understanding that by r_x, r_y, s_x and s_y we mean the functions of (r,s) which form the inverse of the Jacobian, we can express the problem (1.2.1) in (r,s) coordinates as

$$\mathbf{F}\left(r_x\frac{\partial}{\partial r}+s_x\frac{\partial}{\partial s}, r_y\frac{\partial}{\partial r}+s_y\frac{\partial}{\partial s}, \mathbf{u}'\right)=0 \quad \text{for} \quad (r,s)\in[0,1]^2. \tag{1.2.5}$$

If the boundary-value problem involves at most first-order derivatives and r_x, r_y, s_x and s_y are continuous, the problem expressed in (r,s) coordinates is equivalent to the original problem. If the BVP involves nth derivatives, the transformation must satisfy the additional constraints

$$\frac{\partial^{p+q}\mathbf{d}}{\partial r^p \partial s^q} \qquad \text{continuous for all } p+q\le n. \tag{1.2.6}$$

A problem may arise when parts of the boundary of the unit square do not correspond, under the transformation $\mathbf{d}$, to the boundary of $\mathbf{D}$. This occurs when part of the boundary of the unit square corresponds to a branch cut. If all of the required derivatives of $\mathbf{d}$ are continuous across the branch cut, the appropriate periodicity conditions are obvious. For example, if the branch cut is the image of the two sides $r=0$ and $r=1$ of the unit square and the Jacobian of $\mathbf{d}$ extends smoothly to a function periodic in r (with period 1) then the appropriate periodicity condition is that $\mathbf{u}'$ must extend smoothly to a function periodic in r. It may be useful to note in passing that curvilinear grids may also be used for problems, such as the graphical representation of data on a region with a smooth boundary, where no differentiation is involved. For such problems the transformation may not need to be smooth at all; there may be no conditions at all on the Jacobian and its inverse.

If a sufficiently fine uniform rectangular grid on the unit square and an appropriate difference method are used, an accurate discrete solution to the problem (1.2.1) may be obtained. The following second-order central difference approximations may

be used for many problems:

$$
\begin{aligned}
\frac{\partial u}{\partial x} &= r_x D_{0r} u' + s_x D_{0s} u' + O(\Delta r^2, \Delta s^2) \\
\frac{\partial u}{\partial y} &= r_y D_{0r} u' + s_y D_{0s} u' + O(\Delta r^2, \Delta s^2) \\
\frac{\partial^2 u}{\partial x^2} &= (r_x)^2 (D_r^2 u') + 2 r_x s_x (D_r D_s u') + (s_x)^2 (D_s^2 u') \\
&\quad + [r_x (D_{0r} r_x) + s_x (D_{0s} r_x)](D_{0r} u') + [r_x (D_{0r} s_x) + s_x (D_{0s} s_x)](D_{0s} u') \\
&\quad + O(\Delta r^2, \Delta s^2) \\
\frac{\partial^2 u}{\partial x \partial y} &= r_x r_y (D_r^2 u') + (r_y s_x - s_y r_x)(D_r D_s u') + s_x s_y (D_s^2 u') \\
&\quad + [r_x (D_{0r} r_y) + s_x (D_{0s} r_y)](D_{0r} u') + [r_x (D_{0r} s_y) + s_x (D_{0s} s_y)](D_{0s} u') \\
&\quad + O(\Delta r^2, \Delta s^2) \\
\frac{\partial^2 u}{\partial y^2} &= (r_y)^2 (D_r^2 u') + 2 r_y s_y (D_r D_s u') + (s_y)^2 (D_s^2 u') \\
&\quad + [r_y (D_{0r} r_y) + s_y (D_{0s} r_y)](D_{0r} u') + [r_y (D_{0r} s_y) + s_y (D_{0s} s_y)](D_{0s} u') \\
&\quad + O(\Delta r^2, \Delta s^2)
\end{aligned}
\tag{1.2.7}
$$

where

$$
\begin{aligned}
D_{0r} v(r_i, s_j) &= \frac{v(r_{i+1}, s_j) - v(r_{i-1}, s_j)}{2\Delta r} \\
D_{0s} v(r_i, s_j) &= \frac{v(r_i, s_{j+1}) - v(r_i, s_{j-1})}{2\Delta s} \\
D_r^2 v(r_i, s_j) &= \frac{v(r_{i+1}, s_j) - 2v(r_i, s_j) + v(r_{i-1}, s_j)}{\Delta r^2} \\
D_s^2 v(r_i, s_j) &= \frac{v(r_i, s_{j+1}) - 2v(r_i, s_j) + v(r_i, s_{j-1})}{\Delta s^2} \\
\Delta r = \frac{1}{m-1} \quad &\text{and} \quad \Delta s = \frac{1}{n-1}.
\end{aligned}
\tag{1.2.8}
$$

The following second-order one-sided differences may in many instances be used instead of D_{0r} and/or D_{0s} for boundary conditions involving first derivatives:

$$
\begin{aligned}
D_{-r} v(r_i, s_j) &= \frac{2v(r_i, s_j) - 3v(r_{i-1}, s_j) + v(r_{i-2}, s_j)}{\Delta r} \\
D_{+r} v(r_i, s_j) &= \frac{-v(r_{i+2}, s_j) + 3v(r_{i+1}, s_j) - 2v(r_i, s_j)}{\Delta r} \\
D_{-s} v(r_i, s_j) &= \frac{2v(r_i, s_j) - 3v(r_i, s_{j-1}) + v(r_i, s_{j-2})}{\Delta r} \\
D_{+s} v(r_i, s_j) &= \frac{-v(r_i, s_{j+2}) + 3v(r_i, s_{j+1}) - 2v(r_i, s_j)}{\Delta r}.
\end{aligned}
\tag{1.2.9}
$$

Periodicity conditions are easy to discretise on a branch cut that is the image under the transformation **d** of two opposite sides of the unit square and the Jacobian of the transformation extends smoothly to a function periodic in the appropriate coordinate, r or s. Periodicity is represented by the discrete condition that the index i or j is to be interpreted modulo $m-1$ or $n-1$:

$$\begin{aligned} &\text{Periodicity in r}: \quad v(r_i, s_j) \equiv v(r_{i+m-1}, s_j) \\ &\text{Periodicity in s}: \quad v(r_i, s_j) \equiv v(r_i, s_{j+n-1}). \end{aligned} \tag{1.2.10}$$

1.3 Curvilinear Grid Construction

Many methods have been devised to construct a one-to-one transformation from the unit square onto a bounded region **D** in the plane, under which a rectangular grid on the unit square corresponds to a boundary-fitted curvilinear grid on **D**. These methods generally fall into one of three classes: those which construct the transformation as the solution to a PDE (usually elliptic) with boundary conditions specifying that the edges of the unit square are mapped to the boundary of **D**, those which construct the transformation algebraically as a weighted sum of points on the boundary or of functions that describe the boundary (and in some cases other points or functions describing curves interior to **D** also), and those which construct the transformation from commonly known coordinate transformations by the appropriate choice of parameters. The latter methods readily provide transformations for such simple regions as a trapezoid or an annulus, but lack the versatility to handle regions of general shape. More sophisticated examples are the conformal Schwartz-Christoffel transformations for general polygonal regions, but these can be written in closed form only for a small class of regions. Generally, as the shape of the region becomes more complicated, these methods offer diminishing return for the effort of finding an appropriate transformation. Algebraic methods are more versatile in that they can provide transformations for regions whose boundaries need not be described in terms of simple functions. These methods can handle regions of relatively

complex geometry with a minimum of effort and computation. Methods based on PDEs are the most versatile and can be used for regions of quite general shape, but at the expense of the transformation being much slower to compute than one constructed by an algebraic method. Some PDE methods have the advantage of a maximum principle that ensures the resulting transformation is one-to-one. Some algebraic and PDE methods provide control, to a limited degree, over the density of the curvilinear gridlines. The following sections describe the most popular of these methods and some of their strengths and their limitations; Thompson [22] and Eiseman [9] give a more complete survey.

1.3.1 Algebraic Methods of Grid Construction

Given a description of the boundary of $\mathbf{D}$, and in some cases, curves in the interior of $\mathbf{D}$ also, the algebraic methods construct a transformation from the unit square onto $\mathbf{D}$ using various forms of interpolation. These methods are usually described as they apply to regions in three dimensions. However, to avoid notational complexity we shall consider only two-dimensional examples. All of these methods are quick to compute in comparison to the PDE methods. Some of them can handle regions of quite complicated shape. The simplest algebraic method is the shearing transformation

$$\mathbf{d}(r,s) = (1-r)\mathbf{c}_1(s) + r\mathbf{c}_2(s)$$

from the unit square onto the region bounded by two curves $\mathbf{C}_k = \{\mathbf{c}_k(s): 0 \leq s \leq 1\}$ and the line segments joining their endpoints at $s = 0$ and at $s = 1$. This transformation may be used for any region $\mathbf{D}$ whose boundary consists of two line segments joining the endpoints of two curve segments, provided that functions $\mathbf{c}_k$ parametrising the curves $\mathbf{C}_k$ can be found. If the curved part of the boundary of $\mathbf{D}$ is described only by discrete data points, the parametrisations $\mathbf{c}_k$ may be constructed as spline functions through the data points. The main strength of this method is

the speed with which the transformation may be computed, since it requires only two function evaluations. Its main weaknesses are that it can handle only regions with at least two straight sides, one opposite the other, and that the transformation resulting from a region with a convoluted boundary may not be one-to-one; it may have folds where its Jacobian derivative is singular. Most of the other algebraic methods are essentially generalisations of the shearing transformation, designed to overcome one or more of these weaknesses or to provide additional features.

Hermite interpolation is a generalisation of the shearing transformation to allow control over derivatives of the transformation at the curved boundary segments of $\mathbf{D}$. For example,

$$\mathbf{d}(r,s) = (1-3r^2-2r^3)\mathbf{c}_{10}(s) + r^2(3-2r)\mathbf{c}_{20}(s) + r(1-r)^2\mathbf{c}_{11}(s) + r^2(1-r)\mathbf{c}_{21}(s)$$

allows both $\mathbf{d}$ and $\partial\mathbf{d}/\partial r$ to be specified at $r = 0$ and $r = 1$:

$$\mathbf{d}(0,s) = \mathbf{c}_{10}(s), \quad \frac{\partial \mathbf{d}}{\partial r}(0,s) = c_{11}(s)$$
$$\mathbf{d}(1,s) = \mathbf{c}_{20}(s), \quad \frac{\partial \mathbf{d}}{\partial r}(1,s) = c_{21}(s).$$

This control over the derivative at the boundary is useful when coordinate transformations for two adjacent regions must be found such that their Jacobian derivatives are identical along their common boundary. Hermite interpolation suffers the same weaknesses as the shearing transformation.

Another generalisation of the shearing transformation, the "multisurface transformation" proposed by Eiseman [8], provides control over the transformation and its derivative on curves in the interior of $\mathbf{D}$, so that

$$\mathbf{d}(r_k,s) = \mathbf{c}_{k0}(s), \quad \frac{\partial \mathbf{d}}{\partial r}(r_k,s) = \mathbf{c}_{k1}(s)$$

may be specified on arbitrary lines $0 = r_1, r_2, \ldots, r_N = 1$. The multisurface transformation, like Hermite interpolation, is a linear combination of the functions $\mathbf{c}_{k0}$ and $\mathbf{c}_{k1}$ with polynomial coefficients; the coefficients of the polynomials are determined

by a linear system of equations. It maps the unit square onto the region bounded by $\mathbf{C}_1$, $\mathbf{C}_N$ and two curves that pass through the points $\{\mathbf{c}_{k0}(0): k = 1, \ldots, N\}$ and $\{\mathbf{c}_{k1}(1): k = 1, \ldots, N\}$, respectively. It may be used for regions $\mathbf{D}$ of more complicated shape. Another advantage it has over the shearing transformation is additional control over the transformation (and over resolution in the r-direction) in the interior of $\mathbf{D}$. A one-to-one multisurface transformation can be found in many cases where a one-to-one shearing transformation cannot.

Still another generalisation of the shearing transformation is the "Coons patch," [7] or "transfinite interpolation." The simplest formulation of this,

$$\mathbf{d}(r,s) = (1-s)\mathbf{c}_1(r) + s\mathbf{c}_2(r) + (1-r)\mathbf{c}_3(s) + r\mathbf{c}_4(s)$$
$$- (1-s)(1-r)\mathbf{c}_1(0) - (1-s)r\mathbf{c}_2(0) - s(1-r)\mathbf{c}_1(1) - sr\mathbf{c}_2(1),$$

maps the unit square onto the region $\mathbf{D}$ bounded by four curves $\mathbf{C}_k = \{\mathbf{c}_k(s): 0 \leq s \leq 1\}$ whose parametrisations are chosen so that they intersect at $\mathbf{c}_1(0) = \mathbf{c}_3(0)$, $\mathbf{c}_1(1) = \mathbf{c}_4(0)$, $\mathbf{c}_2(0) = \mathbf{c}_3(1)$ and $\mathbf{c}_2(1) = \mathbf{c}_4(1)$. Like the shearing transformation, the simple Coons patch is fast to compute, requiring only four function evaluations. Its advantage is that it can handle a region whose boundary consists of four given curve segments. It has the same trouble with regions that have convoluted boundaries; the resulting transformation may have folds where its Jacobian derivative is singular. A more general formulation of the Coons patch allows $\mathbf{d}(r,s)$ to be specified on more than two curves in each direction,

$$\mathbf{d}(r_i, s) = \mathbf{c}_{1i}(s) \quad \text{for } i = 1, \ldots, M$$
$$\mathbf{d}(r, s_j) = \mathbf{c}_{2j}(s) \quad \text{for } j = 1, \ldots, N,$$

with a specified number of derivatives continuous across these curves. This generalisation can handle regions with more complicated boundaries.

1.3.2 Grid Construction Based on PDEs

PDE methods for construction of coordinate transformations from the unit square onto a region $\mathbf{D}$ are generally more versatile than algebraic methods in

that they can handle regions of more complicated shape and that they offer greater control over resolution and skewness. This versatility comes at the expense of high computational cost. The most popular of these methods construct the transformation as the solution to an elliptic PDE boundary-value problem. Other methods use hyperbolic PDEs (Starius [17], for example) or parabolic PDEs.

The best-known elliptic method for construction of a coordinate transformation is conformal mapping. A conformal map can be found for any simply-connected region $\mathbf{D}$ in two dimensions, as the solution to the Cauchy-Riemann equations

$$\frac{\partial r}{\partial x} = \frac{\partial s}{\partial y}, \quad \frac{\partial s}{\partial x} = -\frac{\partial r}{\partial y}$$

in the interior of $\mathbf{D}$ with boundary geometry $(r, s) = \mathbf{\Psi}(x, y)$ specified, where $\mathbf{\Psi}$ is a function (to be determined) that maps the boundary of $\mathbf{D}$ onto the boundary of the unit square. By differentiating and combining the Cauchy-Riemann equations, we get Laplace's equation $\nabla^2(r, s) = 0$ in the interior of $\mathbf{D}$. With the Cauchy-Riemann equations as additional boundary conditions these may alternatively be used in the interior of $\mathbf{D}$. Conformal maps offer no control over resolution in the interior or on the boundary; gridlines nearby the boundary of $\mathbf{D}$ are always more dense where the boundary is concave than where it is convex. Conformal maps usually have singularities on the boundary of $\mathbf{D}$ where the Jacobian derivative becomes zero or infinite. For example, the conformal map from a circular disc onto the unit square is singular at the points corresponding to the corners of the unit square, and the inverse mapping is singular at the corners of the unit square. Furthermore, conformal mapping methods do not generalise to three dimensions. For these reasons conformal maps are not always appropriate.

As a more useful method for generating coordinate transformations, Winslow [23] proposed using Laplace's equations $\nabla^2(r, s) = 0$, as in the conformal mapping method, but relaxing the boundary conditions. The boundary condition $(r, s) = \mathbf{\Psi}(x, y)$ is imposed, with the function $\mathbf{\Psi}$ specified; the Cauchy-Riemann equations

are not imposed at the boundary. Godunov and Prokipov [10] have shown that a maximum principle guarantees that the Jacobian of the resulting transformation is nonsingular. This approach has the advantage over conformal mapping that it allows some control over the transformation near the boundary. In particular the boundary conditions can be chosen so that corners of $\mathbf{D}$ correspond to corners of the unit square, so that the transformation need not have singuarities. Also, the transformation is faster to compute since it involves solving only a linear problem. This method generalises easily to three dimensions. A disadvantage is that the resulting grid is no longer orthogonal.

The use of elliptic systems for generating coordinate transformations was further generalised by Thompson, Thames and Mastin [21] with the inclusion of forcing terms in the elliptic equations, and branch cuts to allow for multiply-connected regions. The terms f and g in the equations $\nabla^2(r,s) = (f,g)$ allow much greater control over the density of gridlines, not only near the boundary but throughout the interior. Constraints on f and g have been found so that a maximum principle still guarantees that the Jacobian of the transformation is non-singular. The introduction of more than one branch cut, as neccessary for many multiply-connected regions, results in a region $\mathbf{D}$ with more than four corners. In such cases some of the corners of $\mathbf{D}$ cannot correspond to corners of the unit square, and the transformation must have singularities there.

Brackbill and Saltzman [4] generate coordinate systems using a variational formulation. They find the transformation that maximizes a functional $I = I_s + \lambda_v I_v + \lambda_0' I_0'$, where I_s is a global measure of the smoothness of the transformation, I_v is a global measure of resolution and I_0' is a global measure of orthogonality. The Euler equation of this variational problem is an elliptic system of which the system considered by Thompson is a special case. The variational formulation is used to specify the forcing functions in the elliptic problem. This method provides more

systematic global control over the resolution and other properties of the grid. This systematic control makes possible the automatic adaptive generation of grids.

CHAPTER 2

Composite Grids

Two limitations apply to the use of a single curvilinear grid on a region with a curved boundary. One is that even on a region with a fairly simple geometry the density of gridlines will inevitably be highly nonuniform. For example, a curvilinear grid for an oblong region with a narrow waist will inevitably have a much higher density of gridlines in the narrow waist than in some other parts of the region. The other limitation is that if the region **D** is not doubly-connected, any transformation must have a singularity somewhere on the boundary. For example, the Jacobian of any transformation from the unit square onto a disc (a simply-connected region) will be singular at some of the corners of the square; if polar coordinates are used, the Jacobian will be singular all along the side of the square corresponding to the center of the disc; if a non-periodic transformation which maps the corners of the square to points on the boundary of the disc is used, the Jacobian will be singular at all the four corners. Both of these problems can be overcome by the use of composite grids consisting of several overlapping curvilinear grids. Since the component grids of a composite grid may be much simpler than a single curvilinear grid for the same region, many of the methods of curvilinear grid construction discussed in the Chapter 1 are adequate to construct the component grids. In fact, any region may be broken up into enough sub-regions so that the shape of each is simple enough that even the least sophisticated methods may be used to construct the component grids.

2.1 Notation for Composite Grids

We shall consider only composite grids in two dimensions, although all of the concepts extend to three or more dimensions. One-dimensional composite grids may be useful as models for higher-dimensional composite grids in the study of stability and convergence of PDE discretisations, but they have no practical advantage over a single stretched grid for the same interval. So we define composite grids as follows.

Definition 2.1.1 *A composite grid for a compact region* $\mathbf{D}$ *in the plane, on a cylinder or on a torus is a set of* N *component curvilinear grids consisting of continuous transformations* $\mathbf{d}_k$ *from the unit square onto sub-regions* $\mathbf{D}_k$ *chosen such that the union of the* $\mathbf{D}_k$ *contains* $\mathbf{D}$ *and the union of the boundaries of the* $\mathbf{D}_k$ *contains the boundary of* $\mathbf{D}$,

$$\mathbf{D} \subseteq \bigcup_k \mathbf{D}_k \quad \text{and} \quad \partial\mathbf{D} \subseteq \bigcup_k \partial\mathbf{D}_k, \quad \text{where}$$
$$\mathbf{D}_k = \{\mathbf{d}_k(r,s): 0 \le r \le 1, 0 \le s \le 1\}, \quad \text{and}$$

and uniform rectangular grids $\mathbf{G}_k$ *on the unit square,*

$$\mathbf{G}_k = \left\{ \left(\frac{i-1}{m_k - 1}, \frac{j-1}{n_k - 1} \right) : i = 1, \ldots, m_k, \, j = 1, \ldots, n_k \right\},$$

for $1 \le k \le N$.

The component grids of a composite grid are curvilinear grids as defined in the Chapter 1. All of the remarks there about branch cuts due to the transformations and branch cuts on the cylinder or torus apply to these grids. Any region $\mathbf{D}$ may be broken up into enough subregions so that each may have a simple geometry. For example, a multiply-connected region $\mathbf{D}$ in the plane whose boundary consists of n smooth closed curves may be broken up into a rectangular grid on a rectangle that encloses $\mathbf{D}$ and n annular grids each following one of the boundary curves. Those parts of the recangular grid which extend beyond the boundary of $\mathbf{D}$ can be ignored. The transformation used for the rectangular grid introduces no branch cuts. It can be a linear function of (r,s). The transformations used for the annular

grids can be chosen to map one side of the unit square onto the boundary curve, the opposite side of the unit square onto a nearby closed curve in the interior of $\mathbf{D}$ and the other two sides of the unit square onto a branch cut. The Jacobian of each of these transformations extends to a function periodic in the tangential direction and the transformation can be chosen so that its Jacobian is nonsingular and continuous, even across the branch cut.

2.2 PDEs on Composite Grids

Consider a PDE boundary value problem

$$\mathbf{F}\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \mathbf{u}\right) = 0 \quad \text{for} \quad (x,y) \in \mathbf{D} \tag{2.2.1}$$

where $\mathbf{F}$ represents the PDE in the interior of $\mathbf{D}$ and the boundary conditions on the boundary of $\mathbf{D}$, and it is understood that higher derivatives may be involved. For each k we can define a function $\mathbf{u}_k$ which represents $\mathbf{u}$ in cartesian coordinates on the part of the unit square which is mapped by $\mathbf{d}_k$ onto $\mathbf{D}$,

$$\mathbf{u}_k(r,s) = \mathbf{u}(\mathbf{d}_k(r,s)) \quad \text{for} \quad \mathbf{d}_k(r,s) \in \mathbf{D}. \tag{2.2.2}$$

If the Jacobian of the transformation $\mathbf{d}_k$ is non-singular we can use it to express the problem $\mathbf{F}$ in the (r,s) coordinates of each component grid. Let

$$J_k(r,s) = \frac{\partial \mathbf{d}_k}{\partial(r,s)} = \begin{pmatrix} x_r & x_s \\ y_r & y_s \end{pmatrix}. \tag{2.2.3}$$

Then the inverse of $J_k(r,s)$ is

$$J_k^{-1}(r,s) = \frac{1}{x_r y_s - y_r x_s} \begin{pmatrix} y_s & -x_s \\ -y_r & x_r \end{pmatrix} = \begin{pmatrix} r_x & r_y \\ s_x & s_y \end{pmatrix}. \tag{2.2.4}$$

With the understanding that by r_x, r_y, s_x and s_y we mean the functions of (r,s) which form the inverse of the Jacobian, we can express the boundary value problem in (r,s) coordinates as

$$\mathbf{F}\left(r_x\frac{\partial}{\partial r} + s_x\frac{\partial}{\partial s}, r_y\frac{\partial}{\partial r} + s_y\frac{\partial}{\partial s}, \mathbf{u}_k\right) = 0 \quad \text{for} \quad \mathbf{d}_k(r,s) \in \mathbf{D}. \tag{2.2.5}$$

The transformation $\mathbf{d}_k$ must satisfy the smoothness constraints (1.2.6).

Parts of the boundary of the unit square may not correspond, under the transformation $\mathbf{d}_k$, to the boundary of $\mathbf{D}$. A periodicity condition applies on those parts of the boundary of the unit square which correspond to branch cuts. To specify the problem completely we will in general need to impose matching conditions on the remaining parts of the boundary of the unit square. Actually since some parts of the unit square may correspond to the exterior of $\mathbf{D}$ and other parts of the unit square may correspond to parts of $\mathbf{D}$ that are represented on other grids, the PDE and boundary conditions (2.2.5) may apply only on part of the unit square, and the matching conditions need only be applied on the boundary of this part.

We will in general need to apply matching conditions in the regions of overlap between the $\mathbf{D}_k$. In order to do this in a systematic way it helps to be able to order the transformations $\mathbf{d}_k$ so that wherever two or more of the $\mathbf{D}_k$ overlap at the boundary of $\mathbf{D}$, this occurs at the boundary of the $\mathbf{D}_k$ with the highest index k of those involved. Although it is not always possible to order a given set of transformations in this way, it is always possible to choose an appropriate set of transformations that can be so ordered. To help in deciding on an appropriate sequence of transformations, imagine drawing $\mathbf{D}_1$ in the plane, next drawing $\mathbf{D}_2$, erasing that part of $\mathbf{D}_1$ which overlaps $\mathbf{D}_2$, and so on, finally drawing $\mathbf{D}_N$ and erasing those parts of each underlying $\mathbf{D}_k$ which it overlaps. The result will be that the entire region $\mathbf{D}_N$ remains, along with those parts of each underlying $\mathbf{D}_i$ which are not covered by some $\mathbf{D}_j$ with $j > i$. If the sequence was chosen properly, the boundary of $\mathbf{D}$ is composed only of visible parts of the boundaries of the $\mathbf{D}_k$.

Figure 2.1 illustrates how an appropriate sequence $\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3$ of subregions may be chosen for a periodic channel $\mathbf{D}$. First $\mathbf{D}_1$ is chosen to be a rectangle enclosing $\mathbf{D}$. Then $\mathbf{D}_2$ is chosen so that it follows the lower boundary of $\mathbf{D}$ and overlaps $\mathbf{D}_1$. Finally $\mathbf{D}_3$ is chosen so that it follows the upper boundary of $\mathbf{D}$ and overlaps both

$\mathbf{D}_1$ and $\mathbf{D}_2$, with care that it does not overlap the lower boundary of $\mathbf{D}$. If $\mathbf{D}_2$ and $\mathbf{D}_3$ had been chosen in the reverse order, part of the upper boundary of $\mathbf{D}$ would have been hidden and the sequence of subregions would have been inappropriate.

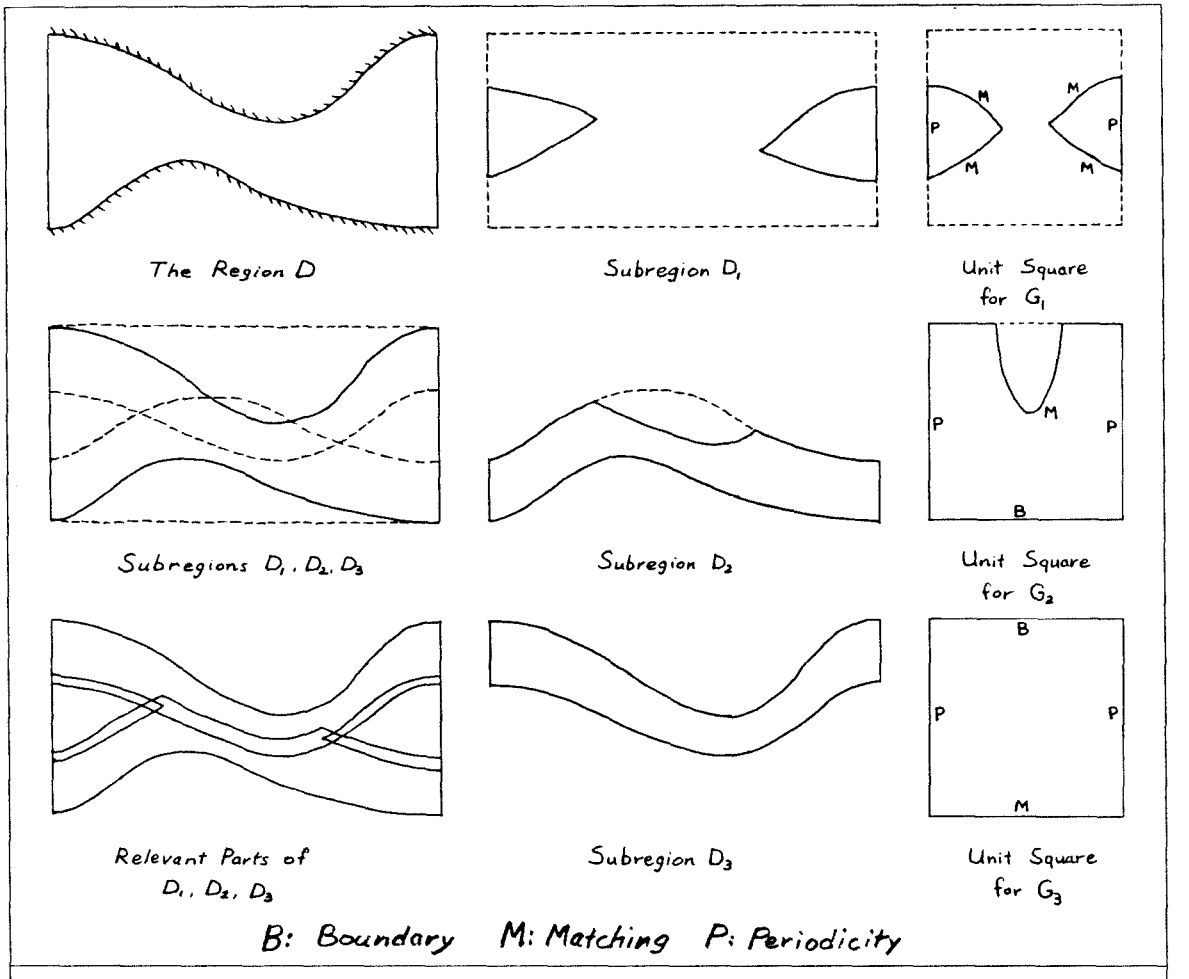


Figure 2.1: Overlapping Subregions of a Periodic Channel

Now we are ready to specify the neccessary matching conditions. First we specify conditions identifying $\mathbf{u}_N$ with $\mathbf{u}_j$ on regions of overlap between $\mathbf{D}_N$ and the underlying regions $\mathbf{D}_j$. Where $\mathbf{D}_N$ overlaps other regions we apply the matching condition $\mathbf{u}_N(r,s) = \mathbf{u}_j(\mathbf{d}_j^{-1}(\mathbf{d}_N(r,s)))$, where j is the highest index of those regions which underlie $\mathbf{D}_N$ at $\mathbf{d}_N(r,s)$. The matching condition is applied only on those edges of $\mathbf{D}_N$ where neither boundary conditions nor periodicity conditions apply.

Depending on the type of PDE involved, some higher derivatives of $\mathbf{u}_N$ and $\mathbf{u}_j$ may also need to be matched in the same way. In this way $\mathbf{D}_N$ is divided into two parts: the part (A) where the PDE and boundary conditions are applied and the part (B) of its boundary where the matching condition is applied. Matching conditions for the underlying regions are similarly specified. Where region $\mathbf{D}_i$ overlaps other regions we apply the matching condition $\mathbf{u}_i(r,s) = \mathbf{u}_j(\mathbf{d}_j^{-1}(\mathbf{d}_i(r,s)))$, where j is the highest index of those regions which overlap or underlie $\mathbf{D}_i$ at $\mathbf{d}_i(r,s)$. The matching condition is applied on a curve (or curves) chosen so as to divide $\mathbf{D}_i$ into three parts: the part (A) where the PDE and boundary condition are applied, the curve or curves (B) where the matching condition is applied, and a part (C) which includes any of $\mathbf{D}_i$ that is exterior to $\mathbf{D}$. Where $i > j$ this curve will lie along edges of $\mathbf{D}_i$ on which neither boundary conditions nor periodicity conditions apply; where $i < j$ the curve lies on that part of $\mathbf{D}_j$ where the PDE and boundary condition would otherwise be applied. By induction we can demonstrate that it is possible to cover $\mathbf{D}$ with overlapping parts of all the $\mathbf{D}_j$ where the PDE is applied, each of which is separated from parts exterior to $\mathbf{D}$ by curves where a matching condition is applied. At this point the problem $\mathbf{F}$ is completely specified in (r,s) coordinates.

Figure 2.1 shows how the curves may be chosen where matching conditions are applied on the three regions $\mathbf{D}_1$, $\mathbf{D}_2$, $\mathbf{D}_3$ of the periodic channel. On $\mathbf{D}_1$ the matching condition $\mathbf{u}_1(r,s) = \mathbf{u}_2(\mathbf{d}_2^{-1}(\mathbf{d}_1(r,s)))$ is applied on the lower halves of curves that separate parts of $\mathbf{D}_1$ at the left and right, where the PDE applies, from a part which extends beyond the boundary or $\mathbf{D}$; the matching condition $\mathbf{u}_1(r,s) = \mathbf{u}_3(\mathbf{d}_3^{-1}(\mathbf{d}_1(r,s)))$ is applied on the upper halves of these curves. On $\mathbf{D}_2$ the matching condition $\mathbf{u}_2(r,s) = \mathbf{u}_1(\mathbf{d}_1^{-1}(\mathbf{d}_2(r,s)))$ is applied on a curve which cuts off part of the top of $\mathbf{D}_2$; the matching condition $\mathbf{u}_2(r,s) = \mathbf{u}_3(\mathbf{d}_3^{-1}(\mathbf{d}_2(r,s)))$ is applied on the remainder of the top edge of $\mathbf{D}_2$. On $\mathbf{D}_3$ the matching condition $\mathbf{u}_3(r,s) = \mathbf{u}_1(\mathbf{d}_1^{-1}(\mathbf{d}_3(r,s)))$ is applied on parts of the bottom edge of $\mathbf{D}_3$ at the

left and right; the matching condition $\mathbf{u}_3(r,s) = \mathbf{u}_2(\mathbf{d}_2^{-1}(\mathbf{d}_3(r,s)))$ is applied on the remainder of the bottom edge.

Each region $\mathbf{D}_k$ is divided into parts of three types: parts (A) where the PDE and boundary condition apply, curves (B) where matching conditions apply, and parts (C) which are either redundant (covered by some other $\mathbf{D}_j$ with $j > k$) or exterior to $\mathbf{D}$. The PDE and boundary conditions (2.2.5) are discretised at all of the gridpoints of $\mathbf{G}_k$ which lie in the part of unit square corresponding to the parts (A) of $\mathbf{D}_k$. For example, the difference approximations (1.2.7) may be appropriate. The discretisation at some of the gridpoints near the curves (B) may involve other gridpoints not in parts (A). The value of $\mathbf{u}$ at these other gridpoints is given by the matching condition, approximated discretely using bivariate interpolation formulas. The set of gridpoints where the discrete matching condition is applied are the discrete analogue of the curves (B) of $\mathbf{D}_k$; gridpoints (A), where the PDE and boundary conditions are discretised, are separated by this set (B) from redundant and/or exterior points (C), where $\mathbf{u}_k$ is not defined.

The matching condition

$$\mathbf{u}_k(r,s) = \mathbf{u}_{k'}(\mathbf{d}_{k'}^{-1}(\mathbf{d}_k(r,s))) \tag{2.2.6}$$

may be approximated at the point $(r_i, s_j) = \left(\frac{i-1}{m_k-1}, \frac{j-1}{n_k-1}\right)$ of grid $\mathbf{G}_k$ using the interpolation formula

$$\mathbf{u}_k(r_i,s_j) = \sum_{(i',j')} \alpha_{kk'}(i,j,i',j')\mathbf{u}_{k'}(r'_{i'}, s'_{j'}) + O(\Delta r'^p, \Delta s'^q) \tag{2.2.7}$$

where $\Delta r' = \frac{1}{m_{k'}-1}$, $\Delta s' = \frac{1}{n_{k'}-1}$, $(r'_{i'}, s'_{j'}) = \left(\frac{i'-1}{m_{k'}-1}, \frac{j'-1}{n_{k'}-1}\right)$ and $\alpha_{kk'}$ is a matrix of coefficients chosen to provide interpolation from grid $\mathbf{G}_{k'}$ onto grid $\mathbf{G}_k$ to the desired order of accuracy. For example, bilinear interpolation of $\mathbf{u}_k(r_i, s_j)$ from grid $\mathbf{G}_{k'}$ takes a linear combination of $\mathbf{u}_{k'}$ at the four gridpoints of $\mathbf{G}_{k'}$ at the corners of a rectangle with sides of length $\Delta r'$ and $\Delta s'$ centered nearest to the point $(r', s') = \mathbf{d}_{k'}^{-1}(\mathbf{d}_k(r_i, s_j))$. The weights in the linear combination are the nonzero coefficients of

$\alpha_{kk'}(i,j,i',j')$ given in Table 2.1, where $\delta r' = (r' - r'_{i'_0})/\Delta r'$, $\delta s' = (s' - s'_{j'_0})/\Delta s'$, and (i'_0, j'_0) are chosen such that $i'_0 - 1 \le r'/\Delta r' \le i'_0$ and $j'_0 - 1 \le s'/\Delta s' \le j'_0$. This gives second-order accuracy ($p = q = 2$) in equation (2.2.7). Biquadratic interpolation of $\mathbf{u}_k(r_i, s_j)$ from grid $\mathbf{G}_{k'}$ takes a linear combination of $\mathbf{u}_{k'}$ at the nine gridpoints of $\mathbf{G}_{k'}$ within a rectangle with sides of length $2\Delta r'$ and $2\Delta s'$ centered nearest to the point (r', s'), with the weights given in Table 2.2, where (i'_0, j'_0) are chosen such that $i'_0 - 1.5 \le r'/\Delta r' \le i'_0 - 0.5$ and $j'_0 - 1.5 \le s'/\Delta s' \le j'_0 - 0.5$. This gives third-order accuracy ($p = q = 3$) in equation (2.2.7).

(i',j')	$\alpha_{kk'}(i,j,i',j')$
(i'_0, j'_0)	$(1-\delta r')(1-\delta s')$
(i'_0+1, j'_0)	$\delta r'(1-\delta s')$
(i'_0, j'_0+1)	$(1-\delta r')\delta s'$
(i'_0+1, j'_0+1)	$\delta r'\delta s'$

Table 2.1: Bilinear Interpolation Weights

(i',j')	$\alpha_{kk'}(i,j,i',j')$
(i'_0-1, j'_0-1)	$\delta r'(1-\delta r')\delta s'(1-\delta s')/4$
(i'_0, j'_0-1)	$-(1+\delta r')(1-\delta r')\delta s'(1-\delta s')/2$
(i'_0+1, j'_0-1)	$-(1+\delta r')\delta r'\delta s'(1-\delta s')/4$
(i'_0-1, j'_0)	$-\delta r'(1-\delta r')(1+\delta s')(1-\delta s')/2$
(i'_0, j'_0)	$(1+\delta r')(1-\delta r')(1+\delta s')(1-\delta s')$
(i'_0+1, j'_0)	$(1+\delta r')\delta r'(1+\delta s')(1-\delta s')/2$
(i'_0-1, j'_0+1)	$-\delta r'(1-\delta r')(1+\delta s')\delta s'/4$
(i'_0, j'_0+1)	$(1+\delta r')(1-\delta r')(1+\delta s')\delta s'/2$
(i'_0+1, j'_0+1)	$(1+\delta r')\delta r'(1+\delta s')\delta s'/4$

Table 2.2: Biquadratic Interpolation Weights

2.3 Composite Grid Construction

The method used here for the construction of composite grids is a generalisation of the method used by B. Kreiss [14] to construct two-component composite

grids for simply-connected regions. Atta and Vadyak [1] also have constructed two-component composite grids, and show that composite grids are useful for three-dimensional problems. Benek et al. [2] have constructed two-component composite grids.

A composite grid for a region $\mathbf{D}$ is a set of curvilinear grids for subregions $\mathbf{D}_k$ whose union contains $\mathbf{D}$ and the union of whose boundaries contains the boundary of $\mathbf{D}$. To construct a composite grid for $\mathbf{D}$ we (1) decide on the sequence of overlapping subregions, (2) construct curves forming the boundary of $\mathbf{D}$ and of the subregions, including any branch cuts, (3) use these curves to construct a curvilinear grid for each $\mathbf{D}_k$, and (4) decide which points on each grid are used for discretisation of the PDE and boundary conditions, which points are interpolated from other grids, and which points are not used at all. The first step is somewhat of an art, and would be difficult to automate. The second and third steps are straightforward, as they require only choosing the parametrisations $\mathbf{c}_i$ of the curves and the transformations $\mathbf{d}_k$ from a small class of available functions and choosing the number of gridlines in each direction. The fourth step would be tedious except that it has been automated. The details of these four steps are given below.

The first step in construction of a composite grid for a region $\mathbf{D}$ is to decide how $\mathbf{D}$ may be broken up into a sequence of N overlapping sub-regions $\mathbf{D}_k$ whose union contains $\mathbf{D}$ and the union of whose boundaries contains the boundary of $\mathbf{D}$, so that the shape of each $\mathbf{D}_k$ is simple enough that a grid can be constructed for each. A single grid will suffice if $\mathbf{D}$ is simple enough. For example, it is easy to construct a grid for a convex region with four corners (something like a rectangle). It is also easy to construct a grid for an annular region each of whose two boundary curves deviates only slightly from a circle. However, in general a composite grid consisting of two or more grids is appropriate and is much easier to construct than a single grid for the same region. For PDE boundary-value problems with boundary layers,

interior layers or other regions where the solution varies rapidly, an appropriate composite grid consists of fine grids in the regions of rapid variation and coarse grids elsewhere. The choice of sub-regions will depend on the application. However, the method outlined in §2.1 applies to a rather general class of problems. In order that matching conditions may be applied, the sequence of subregions must be chosen in accordance with the principles given in §2.2.

The second step is to construct curves forming the boundary of $\mathbf{D}$ and of the subregions. We parametrise each curve $\mathbf{C}_i$ by a smooth function $\mathbf{c}_i$ such that $\mathbf{C}_i = \{\mathbf{c}_i(\xi): 0 \leq \xi \leq 1\}$. These functions may be specified analytically if a formula for the boundary curve shape is known, or by a spline function if only some points on the boundary are known. In order that each transformation $\mathbf{d}_k$ may be smooth and its Jacobian may be nonsingular, we require that the boundary of $\mathbf{D}_k$ and any branch cuts consist of smooth curves intersecting at four corners with included angles smaller than π, and choose $\mathbf{d}_k$ so that the corners of the unit square are mapped to these points. Each side of the unit square is mapped to part of a curve which falls into one of three categories: (1), those curves $\mathbf{C}_i^B$ which form the boundary of $\mathbf{D}$; (2), those curves $\mathbf{C}_i^P$ which have been chosen as branch cuts either of $\mathbf{D}$ (in case $\mathbf{D}$ lies on a cylinder or torus; i.e., is periodic) or of a subregion (in case the subregion is annular); and (3), any other curves $\mathbf{C}_i^I$ in the interior of $\mathbf{D}$ where two or more subregions overlap. A boundary curve $\mathbf{C}_i^B$ or branch cut $\mathbf{C}_i^P$ of $\mathbf{D}$ may form parts of the boundary of more than one of the subregions. In such a case, since the subregions overlap to cover $\mathbf{D}$, those parts of the boundaries of the subregions overlap to cover the curve. Other parts of the boundary of $\mathbf{D}_k$ consist of curves which typically do not form any part of the boundary of another subregion. Branch cuts of $\mathbf{D}$ appear in the plane as pairs of identical curves, shifted with respect to each other by the corresponding period vector. It is important in the discretisation of boundary-value problems that the transformations $\mathbf{d}_k$ from the

unit square constructed using the parametrisations of these curve should have the same periodicity and be smooth across the branch cut. To facilitate this we require that the parametrisations differ by a constant vector, the period vector.

The third step is to construct each component curvilinear grid. We specify the rectangular grid $\mathbf{G}_k$ on the unit square by choosing the numbers n_r and n_s of gridlines in each direction. We construct the transformation $\mathbf{d}_k$ from the unit square onto $\mathbf{D}_k$ using the parametrisations of the curves, segments of which form the boundary of $\mathbf{D}_k$. Each side of the unit square is mapped onto a segment of one of these curves, and each corner of the unit square is mapped onto an intersection of two of these curves. In order that each corner may be well-defined, we require that these intersections not be tangential. Any of the methods discussed in Chapter 1 may be used to construct the transformation. However, since the principal application for composite grids is the discretisation of PDE boundary-value problems, it is usually important to use only those methods which result in a transformation which is smooth, even across branch cuts. Since enough subregions may always be chosen so that the shape of each is very simple, even the simplest methods are usually adequate to construct the transformation $\mathbf{d}_k$. In particular, the "Coons Patch" transformation is guaranteed to be as smooth as the boundary curve parametrisations. To compute it requires only four evaluations of the curve parametrisations, so it is very fast. It is easily modified to allow stretched coordinates for boundary layers and interior layers. For these reasons it is the method of choice in most cases.

The fourth step is to decide which points on each grid are used for discretisation of the PDE and boundary conditions, which points are interpolated from other grids, and which points are not used at all. Points where the PDE or boundary condition may be discretised will be called discretisation points, points which are interpolated from other grids will be called interpolation points, and points which are not used at all will be called exterior points. For simplicity it is assumed that the

PDE and the boundary conditions are both discretised to the same order and that the order is positive and even. If the discretisation is of order p, a square of points (on the rectangular grid) with $p+1$ points on a side is used in the discretisation. The discretisation of the PDE is assumed to be centered, while the discretisation of the boundary condition is assumed to be one-sided. A point may be an discretisation point if all the neighbouring points required for discretisation of the PDE or boundary condition are either discretisation points or interpolation points. Interpolation is assumed to be of order p, where $p > 0$. Interpolation of a point on grid k from grid k' to order p using the formula (2.2.7) requires a square of points on grid k' with p points on a side. The point on grid k may be an interpolation point if all the points in the nearest such square on grid k' are discretisation or interpolation points. An algorithm for finding which points may be discretisation points and which points may be interpolation points, based on these criteria, is given in Chapter 3. This algorithm attempts to minimize the total number of discretisation and interpolation points. When the grids overlap sufficiently and the grids are ordered in accordance with the principles given in §2.2, a PDE boundary-value problem of the given order can be discretised using the discretisation and interpolation points of the resulting composite grid.

2.4 Composite Grids for Multigrid

The multigrid method (see, for example, Brandt [5]) is a fast iterative method for the discrete solution of linear elliptic PDEs. It is especially useful for problems in which the large number points required for an accurate discretisation precludes the use of a direct method to solve the resulting linear system of equations. The multigrid method is usually discussed in the context of a rectangular grid. When an elliptic problem is discretised on a composite grid, it is transformed into a set of elliptic problems on rectangular grids in the (r, s) coordinates of each compo-

nent grid, coupled by interpolation equations. Using this fact, Henshaw [12], [13] (cf. Appendix 2) has implemented a multigrid algorithm for composite grids by incorporating the interpolation equations into the multigrid iteration. This algorithm uses a finite sequence of related composite grids $\mathbf{M}^l$, where $\mathbf{M}^1$ is the grid on which the elliptic problem is to be discretised, and $\mathbf{M}^l$ for $l = 2, 3, \ldots$ are successively coarser composite grids. Corresponding component grids at each multigrid level are constructed using the same transformation $\mathbf{d}_k$. The rectangular grid $\mathbf{G}_k^l$ (on the unit square) of component grid k at multigrid level l contains twice the number of gridlines in each direction as the corresponding rectangular grid $\mathbf{G}_k^{l+1}$ at the next multigrid level. To be precise, the numbers of gridlines in each direction on each grid at each level are related by $m_k^l - 1 = 2(m_k^{l+1} - 1)$ and $n_k^l - 1 = 2(n_k^{l+1} - 1)$.

The multigrid algorithm requires the transfer of data from level l to levels $l-1$ and $l+1$ at each point where the equation or boundary condition is applied. This requirement must be taken into account when deciding which points of each grid at each level are to be used for discretisation of the PDE and boundary conditions, which points are interpolated from other grids, and which points are not used at all. Because of this, the composite grids at all levels must be constructed simultaneously. The one-level algorithm given in Chapter 3 for finding which points may be discretisation points and which may be interpolated generalises to any number of multigrid levels.

CHAPTER 3

Composite Grid Software

We have developed software for the interactive construction of composite grids for compact regions in the plane and for regions with cylindrical or toroidal symmetry. It may be used to specify composite grids consisting of an arbitrary number of grids. A manual for this software is included as Appendix 1. This software has been generalised to construct sequences of composite grids for use with multigrid methods. In addition we have developed general-purpose subroutines for using composite grids in discretising PDE boundary-value problems. These include subroutines for reading a data file containing a description of a composite grid, for calculating the coefficients in the centered second-order discretisation of a linear combination of derivatives on a nine-point stencil, for interpolating data between the grids of a composite grid in the regions of their overlap, and for the second-order discretisation and direct solution of a second-order linear elliptic equation with mixed boundary conditions on a composite grid. These subroutines have been generalised to act on multigrid sets of composite grids, and Henshaw [13] (cf. Appendix 2) has generalised the elliptic solver so that it may use the multigrid algorithm.

3.1 A Program for Composite Grid Construction

We have developed the interactive Fortran program CMPGRD for construction of a composite grid, consisting of as many component grids as neccessary, for any compact region in the plane bounded by piecewise-smooth curves. The region may have cylindrical or toroidal symmetry, i.e., be periodic in one or two directions.

The region may also be multiply-connected; the boundary of the region may consist of any number of distinct closed or periodic curves. CMPGRD may be used to construct the curves forming the boundary of the region, to construct the component grids for subregions bounded by these and other curves, and to decide which points on each grid are used for the discretisation of a PDE and its boundary conditions, which points are interpolated from other grids, and which points are not used at all. The algorithms for each of these steps are given below.

3.1.1 Construction of Curves

There are two methods by which CMPGRD may be used to construct the curves $\mathbf{C}_i$ that bound a region $\mathbf{D}$ and the subregions $\mathbf{D}_k$ which overlap to cover $\mathbf{D}$. They may be specified either analytically by a Fortran subroutine supplied by the user or as spline curves through data points given by the user. Segments of several curves constructed by either method may be used to specify the four parts of the boundary of any subregion $\mathbf{D}_k$ corresponding under the transformation $\mathbf{d}_k$ to the four sides of the unit square, and curves specified by different methods may be used for the same subregion. (CMPGRD is set up so that other methods of curve construction may be incorporated later with a *minimum* of difficulty.) Instructions for using CMPGRD to specify curves are given in §A1.4.2; the computations involved are explained below.

To specify a curve $\mathbf{C}_i$ analytically, the user must provide subroutine CURVE, which, given the curve index i and the arcwise parameter ξ in the range $[0,1]$, returns the value (x,y) and the derivative (x_ξ, y_ξ) of a function $\mathbf{c}_i(\xi)$ which parametrises $\mathbf{C}_i$, and an integer code p_i that indicates how the curve is to be used. General specifications for subroutine CURVE are given in §A1.5.1. For example, if the boundary of the region $\mathbf{D}$ consists of a single closed curve $\mathbf{C}_1$, this curve could be specified analytically by a periodic function $\mathbf{c}_1$ on the interval $[0,1]$; CURVE

should return the value $\mathbf{c}_1(\xi)$ and the derivative of this function, and the integer code $p_1 = -1$ to indicate that this curve is a boundary curve. One of the subregions $\mathbf{D}_1$ used in constructing a composite grid for $\mathbf{D}$ might be an annular region that follows this boundary curve. In that case we would want to find a transformation $\mathbf{d}_1$ that maps one side of the unit square onto the curve $\mathbf{C}_1$, the two adjacent sides onto a branch cut, and the opposite side onto a curve in the interior of $\mathbf{D}$ where $\mathbf{D}_1$ overlaps other subregions. The curve forming the branch cut might be labelled $\mathbf{C}_2$ and the other curve $\mathbf{C}_3$, and these might be specified analytically by functions $\mathbf{c}_2$ and $\mathbf{c}_3$. The branch cut should intersect $\mathbf{C}_1$ at the point $\mathbf{c}_1(0) = \mathbf{c}_1(1)$ and intersect $\mathbf{C}_3$ at $\mathbf{c}_3(0) = \mathbf{c}_3(1)$, so that each side of the unit square may be mapped under the transformation $\mathbf{d}_1$ onto a part of a curve parametrised on a subinterval of [0,1]. In general, every periodic curve should be parametrised on the interval [0,1] so that it intersects a branch cut at each end of the interval. The integer code $p_2 = 2$ is used to indicate that curve $\mathbf{C}_2$ is a branch cut, and that whenever one side of the unit square is mapped by any transformation $\mathbf{d}_k$ onto a segment of this curve, the opposite side of the unit square will be mapped onto the same segment of the same curve. In general, branch cuts occur as pairs of identical curves, so that if $\mathbf{C}_i$ is a branch cut then there must be a corresponding curve $\mathbf{C}_j$, identical to $\mathbf{C}_i$ but possibly shifted with respect to it by a constant vector. This correspondence would be indicated by $p_i = j$ and $p_j = i$, of which $p_2 = 2$ is a trivial case. The integer code $p_3 = 0$ is used to indicate that the closed curve $\mathbf{C}_3$ in the interior of $\mathbf{D}$ is neither a boundary curve nor a branch cut.

CMPGRD may be used to specify a curve as a cubic spline through data points given by the user either in a data file or through the use of interactive graphics. Curves specified in this way are numbered consecutively in the order they are specified, following the highest-numbered curve that was specified analytically by subroutine CURVE. At the time a curve is to be specified, it must be declared as

either a boundary curve, a branch cut or an interpolation curve, and it must be declared whether the curve is to be periodic or not. Each "knot" through which the spline is drawn may be specified either directly by giving its (x, y) coordinates or indirectly by giving the coordinates of a point nearby another curve and taking the knot to be the nearest point on that curve. A nearby point on the nearest curve is determined by a global minimization over a large number (typically 100) of points on each of the previously-specified curves. The nearest point on that curve is determined using a bisection iteration. By this method it is possible to insure that the new curve intersects another particular curve, and that the intersection is at an endpoint of the new curve. In particular, if the new curve was declared to be periodic, the first knot must be a point on a branch cut, and the last knot will be taken to be the corresponding point on the opposite branch cut; if the two branch cuts coincide, the first and last knots are identical and the curve is a closed curve. At least two knots are required to specify a non-periodic curve; a cubic spline curve is constructed which passes through each of the knots in the order they were entered, with zero second derivatives at the end-points. At least four points are needed to specify a closed periodic curve, and at least two points are needed to specify a periodic curve with distinct endpoints; in either case, a cubic spline with periodic boundary conditions is constructed which passes through the knots in the order they were entered, after the last point has been corrected to be the periodic image of the first point. In all cases, the parametrisation of the spline function is chosen so that at the knots, its parameter (independent variable) is proportional to the polygonal arc-length from the initial end-point through the knots. Since only one new curve may be specified at a time, new pairs of distinct branch cuts cannot be specified; any new branch cut is paired with itself.

3.1.2 Construction of Grids

A grid for each subregion $\mathbf{D}_k$ of $\mathbf{D}$ is constructed in several steps. First, the boundary of $\mathbf{D}_k$ is specified as four segments of intersecting curves. Second, a mapping from the unit square onto $\mathbf{D}_k$ is chosen. Third, a stretching function is specified which maps the unit square onto itself, to allow some freedom to resolve boundary and other layers. The transformation $\mathbf{d}_k$ is obtained by first applying the streching, if any, and then the mapping. Fourth, the number of gridlines in each direction is specified. These steps are explained below. Instructions for using CMPGRD to perform these steps are given in §A1.4.3.

The boundary of the subregion $\mathbf{D}_k$ is specified as four segments of intersecting curves; each side of the unit square is mapped under the transformation $\mathbf{d}_k$ onto one of these segments. In order that these intersections be well-defined, only non-tangential intersections may be used to determine them. CMPGRD may be used to specify these segments in terms of data points given by the user either in a data file or through the use of interactive graphics, as follows. First, a curve $\mathbf{C}_i$ is chosen by giving the coordinates of a point nearby that curve. The nearest curve is determined by a global minimization over a large number (typically 100) of points on each of the curves. Then each endpoint of the segment of $\mathbf{C}_i$ is chosen by giving the coordinates of a point nearby the intersection of $\mathbf{C}_i$ with another curve $\mathbf{C}_j$ or nearby an endpoint of $\mathbf{C}_i$. A nearby point on the nearest curve is determined by global minimization over a large number of points on each of the curves. If the given point is nearby the point $\mathbf{c}_i(\xi_0)$ on $\mathbf{C}_i$, the endpoint of the segment is taken to be the endpoint of $\mathbf{C}_i$ nearest to $\mathbf{c}_i(\xi_0)$ in terms of ξ. Otherwise, if the given point is nearby the point $\mathbf{c}_j(\xi_1)$ on $\mathbf{C}_j$ (with $j \neq i$), the endpoint of the segment is taken to be the nearest intersection of $\mathbf{C}_i$ and $\mathbf{C}_j$. This intersection is determined by first finding a point $\mathbf{c}_i(\xi_0)$ on $\mathbf{C}_i$ nearby $\mathbf{c}_j(\xi_1)$ using a global minimization over points on $\mathbf{C}_i$, and then iterating simultaneously on ξ_0 and ξ_1 by Newton's method

until the intersection $\mathbf{c}_i(\xi_{ij}) = \mathbf{c}_j(\xi_{ji})$ is accurately found.

When four curve segments have been specified that together form the boundary of a subregion $\mathbf{D}_k$ of $\mathbf{D}$, we can use these to construct a mapping from the unit square onto $\mathbf{D}_k$. To construct the mapping it is only necessary to associate each side of the unit square with a curve segment and to choose the type of mapping to be constructed from this boundary data. First, a side of the unit square is chosen. Second, a curve segment is chosen by giving the coordinates of a point nearby that segment. The nearest segment is determined by a global minimization over a large number (typically 100) of points on each of the segments. Third, the orientation of the segment is chosen so that each endpoint of the segment corresponds to the appropriate corner of the unit square. CMPGRD checks that the segments associated with each side of the unit square are compatible with each other. In particular, it checks that each corner of the unit square corresponds to the intersection of two curve segments at their endpoints, and that if one side of the unit square corresponds to a segment of a branch cut then the opposite side of the unit square corresponds to the identical segment of the opposite branch cut. When all sides of the unit square are associated with compatible curve segments, parametrisations of the curve segments may be used as boundary data to construct the the mapping from the unit square onto $\mathbf{D}_k$. Although many types of mappings could be constructed, CMPGRD offers only one, a "Coons Patch" [7], also known as transfinte interpolation. . This mapping is a linear combination of points on four segments of curves which form the boundary $\mathbf{D}_k$. To avoid problems with notation, suppose we have numbered the curves so that segments of curves $\mathbf{C}_1$ and $\mathbf{C}_2$ form two opposite sides of the region $\mathbf{D}_k$, and segments of $\mathbf{C}_3$ and $\mathbf{C}_4$ form the other two sides, so that these curves intersect at

$$\begin{aligned} \mathbf{c}_1(\xi_{13}) &= \mathbf{c}_3(\xi_{31}), \quad \mathbf{c}_1(\xi_{14}) = \mathbf{c}_4(\xi_{41}), \\ \mathbf{c}_2(\xi_{23}) &= \mathbf{c}_3(\xi_{32}), \quad \mathbf{c}_2(\xi_{24}) = \mathbf{c}_4(\xi_{42}). \end{aligned} \tag{3.1.1}$$

CMPGRD uses the Coons patch defined by

$$\begin{aligned}(x,y) = {} & (1-u)\mathbf{c}_1(\xi_{13} + (\xi_{14} - \xi_{13})t) + u\mathbf{c}_2(\xi_{23} + (\xi_{24} - \xi_{23})t) \\ & + (1-t)\mathbf{c}_3(\xi_{31} + (\xi_{32} - \xi_{31})u) + t\mathbf{c}_4(\xi_{41} + (\xi_{42} - \xi_{41})u) \\ & - (1-u)(1-t)\mathbf{c}_1(\xi_{13}) - u(1-t)\mathbf{c}_2(\xi_{23}) - (1-u)t\mathbf{c}_1(\xi_{14}) - ut\mathbf{c}_2(\xi_{24})\end{aligned} \tag{3.1.2}$$

CMPGRD is set up so that other methods of grid construction may be incorporated later with a minimum of difficulty. The only restrictions are that the mapping may be evaluated anywhere on the unit square (not just at gridpoints), that it be smooth, and that its Jacobian be invertible wherever $\mathbf{D}_k$ overlaps another subregion of $\mathbf{D}$. If the composite grid is to be used for discretising a PDE boundary-value problem, the Jacobian must be invertible everywhere in the intersection of $\mathbf{D}_k$ and $\mathbf{D}$. CMPGRD checks the Jacobian determinant of the mapping for zeros and changes of sign by evaluating it on a 40 point by 40 point grid on the unit square; if no zeros or changes of sign are found at these points, it searches for zeros or changes of sign nearby the point where the smallest value of the determinant was found.

In order to allow some freedom to construct grids that resolve boundary layers and interior layers, the transformation $\mathbf{d}_k$ from (r,s) coordinates on the unit square to (x,y) coordinates is constructed as the composition of two functions, namely, a "stretching" function from (r,s) coordinates on the unit square to (t,u) coordinates on the unit square (which maps the unit square onto itself), and the mapping function from (t,u) coordinates to (x,y) coordinates as constructed above. Although many different stretching functions could be useful, CMPGRD provides only independent exponential stretching functions in the r and s directions. Each of these is a function of the form

$$r(t) = \frac{t + \sum_{i=1}^{n}[\sigma_i(t) - \sigma_i(0)]}{1 + \sum_{i=1}^{n}[\sigma_i(1) - \sigma_i(0)]} \tag{3.1.3}$$

where

$$\sigma_i(t) = a_i \tanh b_i(t - c_i) \tag{3.1.4}$$

if the problem is not periodic in r, and

$$\sigma_i(t) = a_i \tanh b_i(t - c_i) + \sum_{j=1}^{\infty} a_i[\tanh b_i(t - c_i - j) + \tanh b_i(t - c_i + j)] \quad (3.1.5)$$

if the problem is periodic in r. Using trigonometric identities, the infinite series (3.1.5) may be written as

$$\sigma_i(t) = a_i \tanh b_i(t - c_i) + \sum_{j=1}^{\infty} \frac{a_i \sinh 2b_i(t - c_i)}{\cosh 2b_i(t - c_i) + \cosh 2jb_i}, \quad (3.1.6)$$

which converges quickly and requires only three evaluations of hyperbolic functions (not including the constants $\cosh 2jb_i$). Each term $\sigma_i(t)$ has three free parameters: the weight a_i, the exponent b_i and the location c_i. Each term produces a stretching in the t coordinate centered at $t = c_i$. That is, the image in the (t, u) plane of a uniform grid in the (r, s) plane is a nonuniform rectangular grid with a maximum in density of gridlines for t near each c_i. The density of gridlines in the t-direction at the point (t, u) is proportional to the derivative $r'(t)$, and may be adjusted by varying the parameters a_i, b_i and c_i in each term. In particular, a_i is approximately the ratio of the number of gridlines in the stretched region of the grid near c_i to the total number of gridlines in unstretched regions, while b_i is approximately the ratio of the density of gridlines near c_i to the density of gridlines in unstretched regions. If the problem is periodic in r, an infinite series of correction terms is needed to ensure that the derivative $t'(r)$ of the stretching is periodic. For example, if there is one term in the stretching ($n = 1$), and we choose $a_1 = 1$, $b_1 = 1$ and $c_1 = 0$, the t coordinate is stretched at $t = 0$; if the problem is periodic in r, the t coordinate is equally stretched at $t = 1$. Another sum of the same form expresses s in terms of u, and together these specify $(t, u) = (t(r), u(s))$, where $t(r)$ and $u(s)$ are the inverses of functions given by weighted sums of linear and hyperbolic tangent terms.

3.1.3 Construction of Composite Grids

Once the sequences of regions $\mathbf{D}_k$, transformations $\mathbf{d}_k$ and grids $\mathbf{G}_k$ have been

specified, CMPGRD may be used to decide which points on each grid are used for the discretisation of a PDE and its boundary conditions, which points are interpolated from other grids and which points are not used at all. The algorithm for determining these points is given below. In addition, the (r,s) coordinates (in the unit square of the grid from which it is interpolated) of each interpolated point are computed, so that the coefficients $\alpha_{kk'}$ in the interpolation formula (2.2.7) may be calculated. This information, together with the (x,y) coordinates and Jacobian derivative of each transformation $\mathbf{d}_k$ at the gridpoints of $\mathbf{G}_k$, is sufficient to discretise a PDE boundary-value problem on the composite grid.

Each point on each grid of the composite grid is used either for the discretisation of a PDE or its boundary conditions, or as a point where function values there are interpolated from another grid, or it is not used at all. A gridpoint which may be used for the discretisation of a PDE or its boundary conditions will be called a discretisation point, and a gridpoint where function values are interpolated from another grid will be called an interpolation point. A point which is neither a discretisation point nor an interpolation point will be called an exterior point. A gridpoint may be a discretisation point if each adjacent point used in the discretisation at that point is either a discretisation point or an interpolation point. The points used in the discretisation depend on the order of the discretisation and whether the PDE or the boundary condition is discretised. It is assumed that centered differences are used for the PDE, so that, if second-order differences are used to discretise a second-order differential operator then a nine-point stencil will be required, and if fourth-order differences are used then a 25-point stencil will be required. It is assumed that one-sided differences are used for the boundary conditions, using a stencil of the same size as for the PDE. A gridpoint of $\mathbf{G}_k$ may be an interpolated point, interpolated from $\mathbf{G}_{k'}$, if each point in $\mathbf{G}_{k'}$ from which function values at that point are to be interpolated is either a discretisation point or an interpolation point.

It is assumed that bilinear or biquadratic interpolation is used, so that a square of four or nine points, respectively, of $\mathbf{G}_{k'}$ centered nearest to the (r, s) coordinates (in $\mathbf{G}_{k'}$) of the interpolated point is required for the interpolation.

The first step is to mark each gridpoint on each grid as a discretisation point, so that, unless it is later determined that the point cannot be a discretisation point, it may be used in the discretisation formulae of adjacent discretisation points on the same grid and in the interpolation formulae of interpolation points on other grids.

The second step is to mark non-boundary points that lie close to a boundary side of another grid as exterior points. (Boundary points and boundary sides are those gridpoints and sides of the unit square which are mapped onto points and segments, respectively, of curves which were declared to be boundary curves.) This is done so that if a grid covers a hole in a multiply-connected region, then those of its gridpoints in the hole will be marked as exterior, as they should be.

The third step is to find which grid, if any, each point on each grid can be interpolated from. Starting with the last (highest) grid and working down to the first (lowest) grid, each gridpoint on each $\mathbf{G}_k$ is examined to find the highest $\mathbf{G}_{k'}$ with $k' > k$, if any, from which it can be interpolated. For each side of $\mathbf{G}_k$, each point which is neither a periodic point (a point on a side of the unit square which is mapped onto a branch cut) nor a boundary point, and cannot be interpolated from a higher grid, is then examined to find the highest $\mathbf{G}_{k'}$ with $k' < k$, if any, from which it can be interpolated. The points used in these latter interpolations are marked as needed for interpolation, so that they may not be later marked exterior. The end result is that each point that can be interpolated from a higher grid is marked as interpolated from the highest such grid, each point on each non-boundary, non-branch-cut side of each grid which can only be interpolated from a lower grid is marked as interpolated from the highest such grid, and each point used in the interpolation of a point on a higher grid is marked as needed for interpolation.

The fourth step is to examine each remaining discretisation point to see if it can really be a discretisation point. Each grid is swept to find any point which is currently marked as a discretisation point but some of whose neighbours, required for discretisation, are neither discretisation points nor interpolated points. Such points are then marked as exterior points. If any of these points has been marked as needed for interpolation, the composite grid will be useless and CMPGRD gives an error message to that effect. Each grid is swept repeatedly until no more points are found which must be changed to exterior points. The end result is that each point marked as a discretisation point really can be a discretisation point, but that some of the points marked as interpolated points can no longer be interpolated. However, provided there is sufficient overlap between the grids, all the interpolation points needed for discretisation at the remaining discretisation points can still be interpolated.

The fifth step is to examine each interpolated point to see if it is needed for the discretisation of any discretisation point or has been marked as needed for interpolation. Starting with the lowest grid and proceeding to the highest, each interpolated point on each grid is examined. Those interpolated points which are not needed are marked as exterior points. Those points on higher grids required to interpolate the remaining interpolation points are marked as needed for interpolation. The end result is that all of the remaining interpolation points can really be interpolated, provided there is sufficient overlap between the grids, and that each point needed for interpolation of some point on another grid is marked as such.

At this point all the points on all the grids are marked truthfully as to whether they may be discretisation points, interpolation points or neither. However, when there is a choice, it may be preferable that a point be a discretisation point rather than an interpolation point, since the system of interpolation equations may be better-conditioned if there are fewer interpolation points, and hence fewer equations.

The sixth step is to examine each interpolation point to see whether it could just as well be a discretisation point, and if so, then mark it as such. The end result is a composite grid with fewer interpolation points.

The seventh and final step is to output to a file all the information about the composite grid relevant to discretising a PDE boundary-value problem, namely: the coordinate bounds of the region $\mathbf{D}$, defining a bounding rectangle to help with plotting; the number of grids; the number of gridlines in each direction on each grid; a flag array indicating for each point on each grid whether the point is a discretisation point, an interpolation point or neither; the boundary conditions on each side of each grid, indicating also which curve the side is mapped to; a list of interpolated points on each grid, indicating the grid from which it is interpolated and the (r, s) coordinates of the point on that grid; the (x, y) coordinates and the Jacobian derivative at each point on each grid. This file may be read, with the help of a utility subroutine described in the next section, by a program written to discretise a PDE boundary-value problem.

The above algorithm spends most of its time in computing the (r, s) coordinates on one $\mathbf{G}_{k'}$ of points on another $\mathbf{G}_k$. CMPGRD stores the coordinates

$$(x, y) = \mathbf{d}_k(r_i, s_j) = \mathbf{d}_k\left(\frac{i-1}{m_k - 1}, \frac{j-1}{n_k - 1}\right)$$

of all the points on all grids, but must calculate the inverses of these transformations

$$(r', s') = \mathbf{d}_{k'}^{-1}(x, y)$$

to determine, for example, whether the gridpoint (r_i, s_j) can be interpolated from $\mathbf{G}_{k'}$. This function inversion is performed in several steps. First, it is determined whether (x, y) lies within a previously-calculated rectangle which bounds $\mathbf{D}_{k'}$. If not, then the inversion does not proceed. Second, beginning with an arbitrary gridpoint (i', j') on $\mathbf{G}_{k'}$, a sequence of adjacent gridpoints of $\mathbf{G}_{k'}$ is followed which finds a local minimum of $|x - x'| + |y - y'|$. Third, if this local minimum occurs on

the boundary of $\mathbf{G}_{k'}$, then $|x-x'|+|y-y'|$ is minimized globally over the boundary of $\mathbf{G}_{k'}$ and the second step is repeated. At this point we have found the nearest point (i',j') on $\mathbf{G}_{k'}$ in the one norm. Fourth, Newton's method is used to solve the equation $\mathbf{d}_{k'}(r',s') = (x,y)$ starting from the coordinates of the point (i',j'), taking periodicity into account when neccessary. If the Newton interations take (r',s') outside the unit square (beyond a small tolerance), the inversion does not proceed.

3.2 Subroutines for Using Composite Grids

We have developed several general-purpose subroutines for using composite grids in discretising PDE boundary-value problems. These include subroutine CGDATA for reading a data file containing a description of a composite grid, subroutine DISCRT for calculating the coefficients in the centered second-order discretisation of a linear combination of derivatives on a nine-point stencil, subroutine INTERP for interpolating data between the grids of a composite grid in the regions of their overlap, and subroutine SOLVER for the second-order discretisation and direct solution of a second-order linear elliptic equation with mixed boundary conditions on a composite grid. The use of these subroutines is outlined below.

CGDATA is a Fortran subroutine that reads a file containing the description of a composite grid and stores it in a form that can be used by other composite grid subroutines.

DISCRT is a Fortran subroutine that, given a point (i,j) on $\mathbf{G}_k$ and a set of coefficients c_{xx}, c_{xy}, c_{yy}, c_x, c_y and c_c, computes the weights for the nine points in the centered second-order discretisation of the operator

$$L \equiv c_{xx}\frac{\partial^2}{\partial x^2} + c_{xy}\frac{\partial^2}{\partial x \partial y} + c_{yy}\frac{\partial^2}{\partial y^2} + c_x\frac{\partial}{\partial x} + c_y\frac{\partial}{\partial y} + c_c \qquad (3.2.1)$$

on $\mathbf{G}_k$ using the difference approximations (1.2.7) and (1.2.8). This subroutine is meant as a prototype to help in discretising PDE problems.

INTERP is a Fortran subroutine that, given a function defined on the discretisation points of a composite grid, computes interpolated values of the function at the interpolation points using bilinear or biquadratic interpolation formulae. If the interpolation formulae for any of the interpolation points involve interpolation points on other grids, then the interpolation formulae become a coupled linear system of equations. INTERP factors this sparse system using subroutines from the Yale Sparse Matrix Package (YSMP) [11].

SOLVER is a Fortran subroutine that computes the second-order discretisation and direct solution of a second-order linear elliptic equation

$$c_{xx}\frac{\partial^2 u}{\partial x^2} + c_{xy}\frac{\partial^2 u}{\partial x \partial y} + c_{yy}\frac{\partial^2 u}{\partial y^2} + c_x\frac{\partial u}{\partial x} + c_y\frac{\partial u}{\partial y} + c_c u = f \qquad (3.2.2)$$

with oblique mixed boundary condition

$$b_x\frac{\partial u}{\partial x} + b_y\frac{\partial u}{\partial y} + b_c u = g \qquad (3.2.3)$$

using the difference approximations (1.2.7), (1.2.8) and (1.2.9), and bilinear or biquadratic interpolation. SOLVER factors this sparse system using subroutines from the Yale Sparse Matrix Package (YSMP) [11].

CHAPTER 4

Applications of Composite Grids

Composite grids are useful in discretising a wide variety of PDE boundary-value problems by finite-difference methods. The primary advantage of using a composite grid rather than a single grid is the flexibility it allows in discretising problems on regions of complicated geometry and problems with boundary layers and internal layers. The high degree of control over the density of gridlines in a composite grid permits accurate discretisation of problems using in some cases only a fraction of the number of gridpoints a single grid would require. We have developed general-purpose subroutines for using composite grids in discretising PDE boundary-value problems. The discretisations of an elliptic boundary value problem with an internal layer and of a supersonic flow problem with multiple shocks are given as examples of the uses of composite grids and the capabilities of the software.

4.1 An Elliptic Problem With an Internal Layer

To demonstrate how effectively composite grids may be used in PDE boundary-value problems with layers of rapid transition, we consider here a second-order elliptic equation with Dirichlet boundary condition. The coefficients in the equation are chosen so that the solution will have an interior layer, and the forcing terms and boundary data are chosen so that the exact solution is known. Numerical results show that when the internal layer is very sharp, a composite grid can be used to calculate an accurate solution, while a single rectangular grid cannot; the number of gridpoints required to resolve the solution on a single grid is prohibitive.

We consider an elliptic boundary-value problem of the form

$$a(x,y)\nabla^2 w - w = f(x,y) \tag{4.1.1}$$

inside the square $\mathbf{D} = [-1,1]^2$ with Dirichlet boundary condition $w(x,y) = g(x,y)$, where $a(x,y) = O(1)$, $f(x,y) = O(1)$ and $g(x,y) = O(1)$, and $a(x,y)$ is bounded away from zero independently of ε except near the circle of radius one half, where $a(x,y) = O(\varepsilon^2)$. In particular, we choose

$$a(x,y) = \left(\frac{1}{Q'(P(x,y))}\right)^2, \quad \text{where}$$

$$Q(p) = \frac{p + \tanh\frac{2p-1}{2\varepsilon} + \tanh\frac{2p+1}{2\varepsilon}}{1 + \tanh\frac{1}{2\varepsilon} + \tanh\frac{3}{2\varepsilon}} \quad \text{and} \quad P(x,y) = \sqrt{x^2+y^2}.$$

For positive values of ε, $Q(r)$ is a smooth, odd function of r, so $a(x,y)$ is a smooth function of x and y. For small, positive ε, $a(x,y)$ has approximately the value 9 over all of the unit square, except near the circle of radius one half where it drops to approximately $9\varepsilon^2$. We choose the forcing function and boundary data so that the exact solution is $w = \cos\pi Q(P(x,y))$. In particular we choose

$$f(x,y) = -(1+\pi^2)\cos\pi Q - \left[\frac{Q''}{(Q')^2} + \frac{1}{PQ'}\right]\pi\sin\pi Q \quad \text{and}$$

$$g(x,y) = \cos\pi Q, \quad \text{where} \quad Q = Q(P) \quad \text{and} \quad P = P(x,y).$$

For small, non-zero values of ε, $w(x,y)$ and $f(x,y)$ are smooth everywhere except for $O(1)$ transitions across a layer of width $O(\varepsilon)$ near the circle of radius one half, and $g(x,y)$ is smooth everywhere on the boundary of $\mathbf{D}$. In the limit as $\varepsilon \to 0$ $w(x,y)$ tends to the function which is one inside the circle of radius one half and zero on the rest of $\mathbf{D}$.

In order to resolve the solution $w(x,y)$ and the coefficient function $f(x,y)$ on a uniform rectangular grid on $\mathbf{D}$, the numbers n_x and n_y of gridlines in each direction must be taken large enough so that the gridline spacing is of the order of the layer width or smaller. That is,

$$\Delta x = \frac{2}{n_x - 1} = O(\varepsilon), \quad \Delta y = \frac{2}{n_y - 1} = O(\varepsilon),$$

and the total number of gridpoints is $N_1 = n_x n_y = O(\varepsilon^{-2})$. For example, if we take $\Delta x = \Delta y = \frac{\varepsilon}{10}$ and $\varepsilon = \frac{1}{40}$ then the total number of gridpoints is $N_1 = 1.6 \times 10^5$. This number of gridpoints would be needed in order to discretise the problem using second-order centered differences and obtain an accurate solution.

We can construct a composite grid, consisting of a radially stretched (nonuniform) annular grid covering part of $\mathbf{D}$ that includes the layer and a uniform square grid covering the rest of $\mathbf{D}$, which resolves $w(x,y)$ and $f(x,y)$ using a number of gridpoints which is independent of the layer width ε. We construct the composite grid shown in Figure 4.1 using the two transformations

$$\mathbf{d}_1(r,s) = (2r-1, 2s-1)$$

from the unit square $[0,1]^2$ onto $\mathbf{D}_1 = \mathbf{D}$ and

$$\mathbf{d}_2(r,s) = (0.2 + 0.6T(r))(\cos 2\pi s, \sin 2\pi s)$$

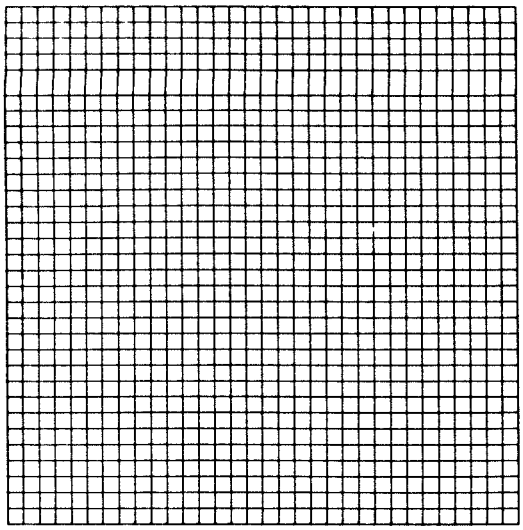
from the unit square onto the annulus $\mathbf{D}_2 = \{(x,y): 0.2 \leq P(x,y) \leq 0.8\}$, where $T(r)$ is the inverse of the function

$$R(t) = \frac{t + \frac{1}{2}(\tanh \frac{2t-1}{2\varepsilon} + \tanh \frac{1}{2\varepsilon})}{1 + \tanh \frac{1}{2\varepsilon}}.$$

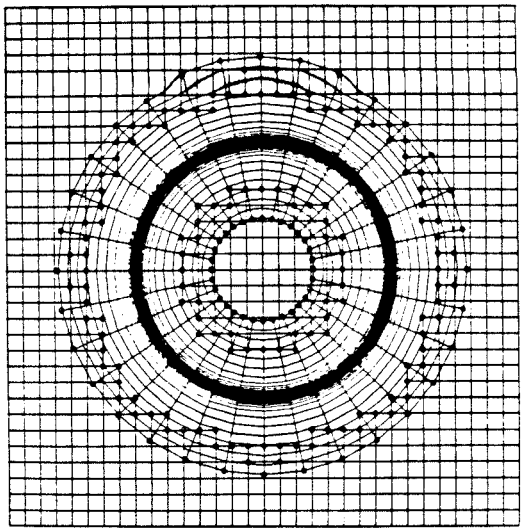
The radial stretching function $R(t)$ concentrates gridlines in a layer of width ε around the circle of radius one half. Although a variety of functions could have been used to achieve the same result, this particular $R(t)$ was chosen because CMPGRD readily provides stretching functions of this form.

In the new coordinates (r,s) given by the transformations $\mathbf{d}_1$ and $\mathbf{d}_2$ the solution has no layer of rapid transition; the problem may be discretised and an accurate solution obtained using a number of gridpoints which is independent of ε. In the scaled rectangular coordinates given by the transformation $\mathbf{d}_1$, equation 4.1.1 becomes

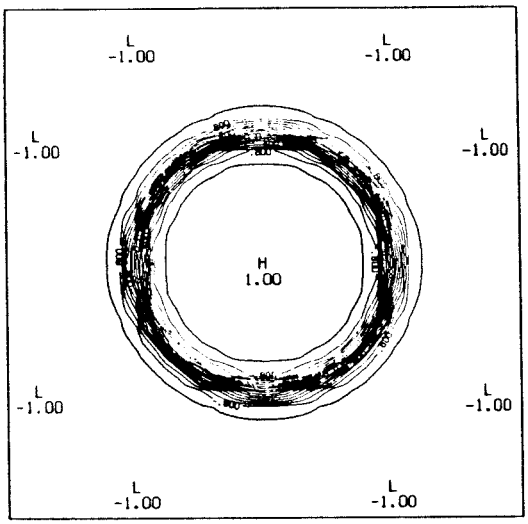
$$\left(\frac{1}{2Q'}\right)^2 \left(\frac{\partial^2}{\partial r^2} + \frac{\partial^2}{\partial s^2}\right) w_1 - w_1 = f(\mathbf{d}_1(r,s)), \qquad (4.1.2)$$



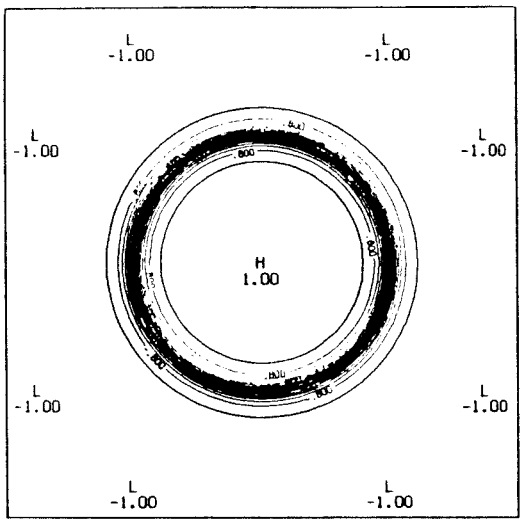
Computational Grid



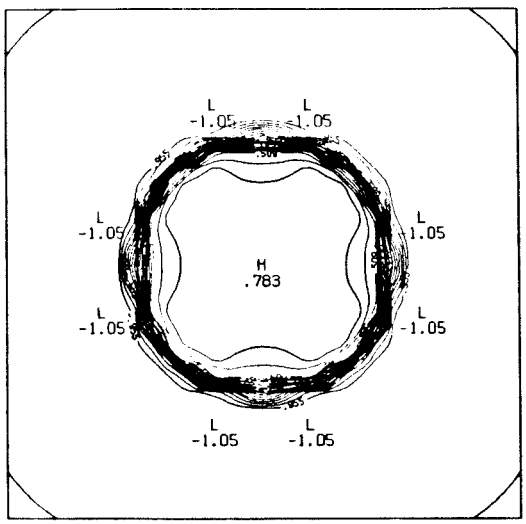
Computational Grid



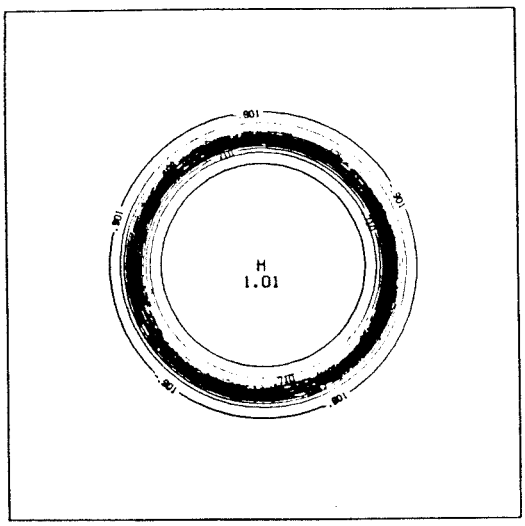
Exact solution



Exact solution



Calculated solution



Calculated solution

Fig. 4.1: Comparison of a Single Rectangular Grid and a Composite Grid for an Elliptic BVP with an Internal Layer

where $w_1(r,s) = w(\mathbf{d}_1(r,s))$. The coefficients of both $\partial^2 w_1/\partial r^2$ and $\partial^2 w_1/\partial s^2$ in equation 4.1.2 are $O(1)$ and bounded away from zero independently of ε, and the coefficients, the forcing function and the boundary values are $O(1)$ and smooth in the part of $\mathbf{D}$ where the square grid is used. Therefore, the number of gridpoints needed to accurately discretise the problem on this part of $\mathbf{D}$ is independent of ε. In the stretched polar coordinates provided by the transformation $\mathbf{d}_2$, equation 4.1.1 becomes

$$\left(\frac{R'}{.6Q'}\right)^2 \left\{\frac{\partial^2 w_2}{\partial r^2} + \left[\frac{R''}{.6(R')^2} + \frac{1}{.6PR'}\right]\frac{\partial w_2}{\partial r}\right\} + \left(\frac{1}{2\pi PQ'}\right)^2 \frac{\partial^2 w_2}{\partial s^2} - w_2 = f(\mathbf{d}_2(r,s)), \tag{4.1.3}$$

where $w_2(r,s) = w(\mathbf{d}_2(r,s))$. The coefficient of $\partial^2 w_2/\partial r^2$ in equation 4.1.3 is $O(1)$ and bounded away from zero independently of ε, and the coefficients, the forcing function and the boundary values are $O(1)$ and smooth for all $(r,s) \in [0,1]^2$. Therefore the number of gridlines in the r-direction needed to accurately discretise the problem on this part of $\mathbf{D}$ is independent of ε. Since we know the solution is independent of s, the number of gridlines in the s-direction does not affect the accuracy. Hence, the number of gridpoints needed is independent of ε.

Table 4.1 gives a comparison of numerical results for this problem for ε in the range of $\frac{1}{10}$ to $\frac{1}{40}$ on a single rectangular grid and on appropriately stretched composite grids. It shows that a fixed number of gridpoints may be used to accurately solve the problem with a composite grid, but not with a singular rectangular grid.

ε	Single Grid	Composite Grid
∞	.0062	
1/10	.0332	.0112
1/20	.4190	.0196
1/40		.0733

Table 4.1: Maximum Error

We can discretise the problem using a number of gridpoints independent of ε

in the above example only because the ratio R'/Q' is $O(1)$ even where the solution has a layer of rapid transition. That is, we were able to choose a function $R(t)$ to specify a transformation of coordinates so that (1) the gridlines are closely spaced only in the layer where Q' is small, and (2) the density of gridlines in the layer is high enough to accurately resolve the PDE coefficients and the solution on the grid. In more general problems with layers this is possible only when the location and width scale of the layers are accurately known. However, even for a problem with a layer whose location is known only approximately, a composite grid can be used to accurately discretise the problem with significantly fewer gridpoints than a single grid would require.

4.2 A Supersonic Flow Problem

A typical feature of supersonic flow is the formation of shocks. In two dimensions these are curves across which some component of the flow is discontinuous. In general these curves may intersect each other and they may appear, move and disappear as the flow evolves. Wherever the flow is smooth, it may be modelled by a nonlinear hyperbolic system. However, such a system cannot model shocks unless, for example, it is regarded as the inviscid limit of a parabolic system, or some special rules are applied wherever shocks appear. We shall take the former approach, which has the advantage of being straightforward to apply and the disadvantage of approximating shocks as layers of rapid transition instead of discontinuities.

Composite grids are well suited to the discretisation of steady supersonic flow problems. By allowing the use of fine grids in the regions of shocks and relatively coarse grids where the solution is smooth they significantly reduce the total number of gridpoints needed to obtain an accurate solution. The position of the shocks cannot in general be determined a priori, so a composite grid which will resolve these shocks cannot initially be found. However, an iterative procedure may be used

to construct such a grid and use it to calculate an accurate steady solution. Since automatic refinement of composite grids is not currently available, this iteration requires considerable human intervention. This approach is practical because only a few iterations are needed in order to construct a composite grid on which the solution is resolved. This approach is impractical for time-dependent solutions with moving shocks since a new composite grid may need to be constructed after every few timesteps.

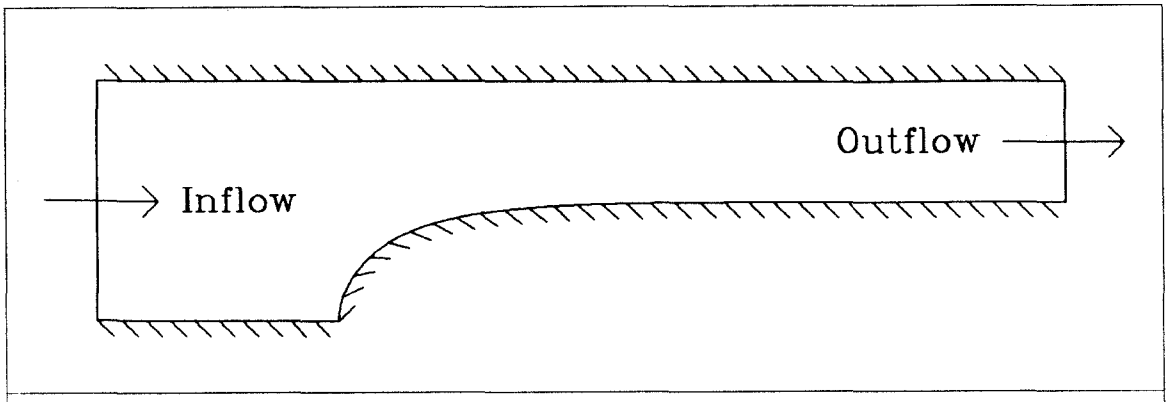


Figure 4.2: A Duct With a Step

To demonstrate how composite grids may be used to discretise a supersonic flow problem with a steady solution, we look at the problem of supersonic flow in a duct with a step. This problem is similar to one considered by Woodward and Colella [24]. In particular we consider a duct extending from $-\pi$ to 3π in the x-direction and from 0 to π in the y-direction, with a curved step extending from $(0,0)$ where it intersects the bottom of the duct at right angles, to a point near $(3\pi, \pi/2)$ at the downstream end of the duct. The shape of the step is described by

$$x + iy = \log(1 + is) \qquad \text{for} \qquad s \geq 0.$$

We consider the case of supersonic inflow so that we may specify all components of the flow at the upstream end of the duct. In particular we specify $\rho = 1.4$, $u = 3.0$, $v = 0.0$ and $p = 0.2$ at $x = -\pi$. At the downstream end the step narrows the duct

to approximately half its full height, so we may anticipate that the outflow velocity is higher than the inflow velocity. We assume that the flow at the downstream end is supersonic and we do not specify any component of the flow at outflow. We specify that there is no normal flow at solid boundaries. With these boundary conditions the flow evolves to a steady solution independent of the initial condition. The main feature of this steady flow is a shock intersecting the bottom wall of the duct upstream of the step and extending to the top wall of the duct, where it is reflected. The reflected shock becomes weaker toward the downstream end and reflects once more from the step before it passes out of the duct.

The Euler equations for compressible flow in two dimensions are

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = 0 \tag{4.2.1}$$

where

$$\mathbf{u} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ e \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ \rho uv \\ u(p+e) \end{pmatrix}, \qquad \mathbf{G} = \begin{pmatrix} \rho v \\ \rho uv \\ p + \rho v^2 \\ v(p+e) \end{pmatrix}, \tag{4.2.2}$$

ρ = density, p = static pressure, (u, v) = velocity and e = internal energy related to the pressure by the equation of state $e = p/(\gamma - 1) + \frac{1}{2}\rho(u^2 + v^2)$. We would like to use the Euler equations 4.2.1 to model supersonic flow in a duct with a step. However, several shocks develop in this flow, and the corresponding solution of the Euler equations becomes discontinuous. Once discontinuities develop in the solution, these equations can no longer be used. To circumvent this difficulty, we add a dissipation term

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \varepsilon \nabla^2 \mathbf{u} \tag{4.2.3}$$

and consider the limit $\varepsilon \to 0$. For small ε the dissipative term has almost no effect where $\mathbf{u}$ is smooth, but acts to smooth out shocks, where $\mathbf{u}$ varies rapidly in space. To see this, consider the case of a steady solution independent of y, and suppose it has a layer of width δ that approximates a shock. That is, $|d\mathbf{u}/dx| = O(|\mathbf{u}|/\delta) =$

$O(\delta^{-1})$ and $|d\mathbf{F}/dx| = O(|\mathbf{F}|/\delta) = O(\delta^{-1})$ in the layer. This solution satisfies the equation $d\mathbf{F}/dx = \varepsilon d^2\mathbf{u}/dx^2$ so in the layer we have $|d\mathbf{u}/dx| = O(|\mathbf{F}|/\varepsilon) = O(\varepsilon^{-1})$. This shows that shocks are smoothed out to layers of width ε.

The parabolic system of equations 4.2.3 requires boundary data for all components of $\mathbf{u}$ on the entire boundary. The extra boundary conditions are chosen so that they do not introduce boundary layers into the solution where there were none in the solution to the Euler equations 4.2.1. In particular, we bound the normal derivative of each component for which we have no other boundary condition. To bound the normal derivative of a component of the solution at the boundary, we set the value of the component at the boundary points of each grid equal to its value on the next line of gridpoints in the interior on the same grid. The boundary conditions we use are given in table 4.2.

At Inflow	At Outflow	On Walls
$\rho = 1.4$	$\partial\rho/\partial n$ bounded	$\partial\rho/\partial n$ bounded
$u = 3.0$	$\partial(u,v)/\partial n$ bounded	$(u,v)\cdot\mathbf{n} = 0$
$v = 0.0$	$\partial v/\partial n$ bounded	$\partial((u,v)\cdot\mathbf{t})/\partial n$ bounded
$p = 0.2$	$\partial e/\partial n$ bounded	$\partial e/\partial n$ bounded

Table 4.2: Boundary Conditions for Equation 4.2.3

We discretise the parabolic system 4.2.3 on a composite grid using leap-frog time differencing with lagged dissipation and centered second-order differences in (r,s) on each grid. We use this time-differencing as a relaxation method to find a steady solution. Biquadratic interpolation is used in the regions of overlap between grids. An accurate solution can be obtained if the grids are fine enough to resolve the shock layers smoothly. If the positions of the layers are unknown then fine grids must be used everywhere and the total number of gridpoints will be $O(\varepsilon^{-2})$. However, if the positions of the layers are known a priori then we can construct a composite grid with fine grids only where they are needed. This greatly reduces the

total number of gridpoints.

We use an iterative procedure to calculate the steady solution to 4.2.3 in the limit $\varepsilon \to 0$. At the first step in the iteration we take a modest value for ε and a reasonable initial condition such as the potential flow

$$\rho = 1.4$$

$$u - iv = 3 + \frac{3}{1 - 2e^{-(x+iy)}}$$

$$p = 0.2$$

chosen to satisfy the boundary conditions. Using a composite grid composed of only coarse grids we run the problem until the solution approaches a steady state. For each subsequent step of the iteration we construct a new composite grid composed of two coarse grids and one or more fine grids in the regions where the steady solution from the previous step was least accurately resolved. Then we take a smaller value for ε, interpolate the final state from the previous step onto the new composite grid as an initial condition and run the problem again until the solution approaches a new steady state. These steps are repeated until a target value of ε is reached. In particular, figures 4.3, 4.4 and 4.5 show the composite grids used and the steady states obtained with $\varepsilon = 0.5$, $\varepsilon = 0.02$ and $\varepsilon = 0.015$, respectively.

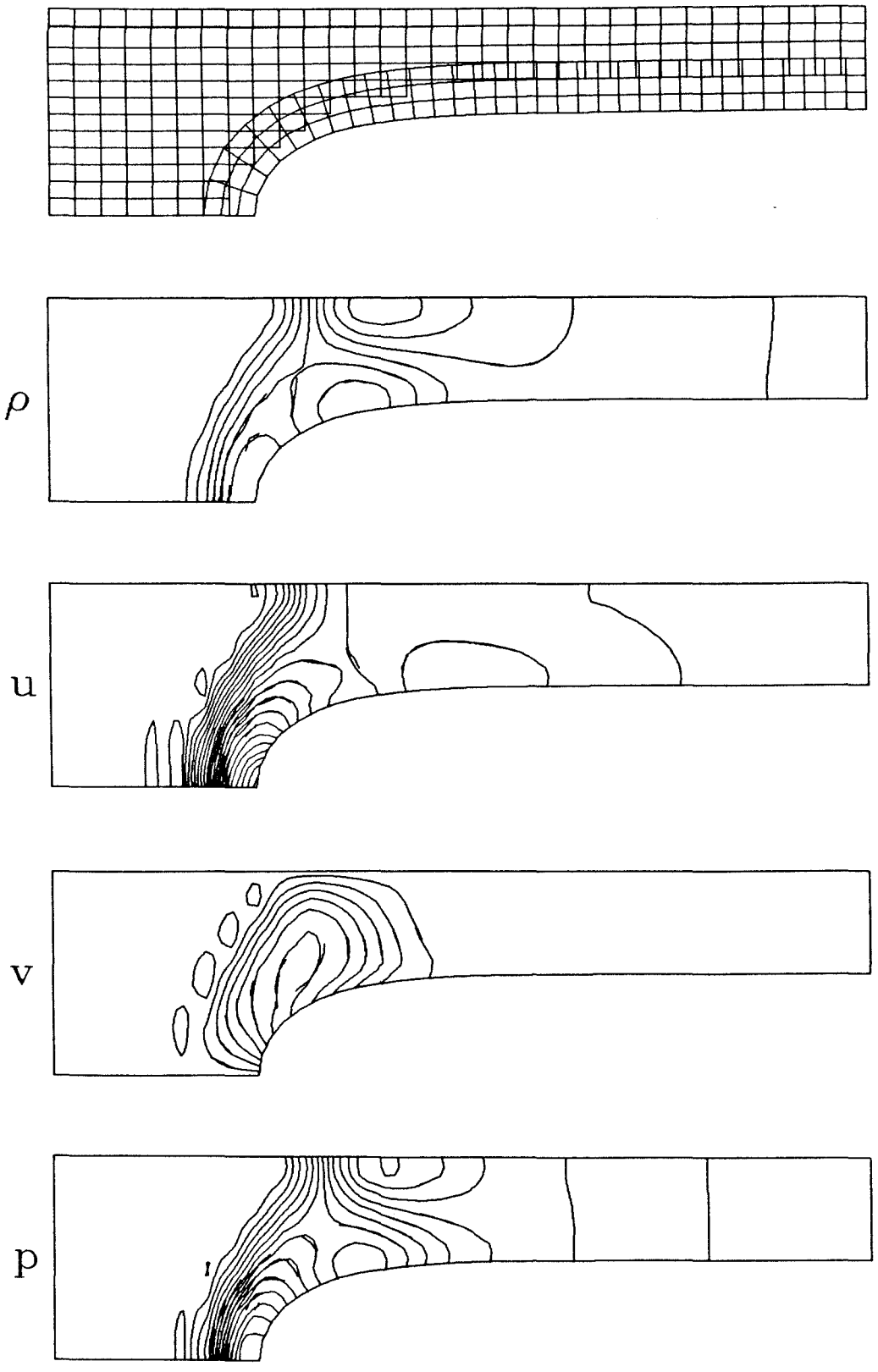


Fig. 4.3: Composite Grid and Steady State for the First Step (with $\varepsilon = 0.5$)

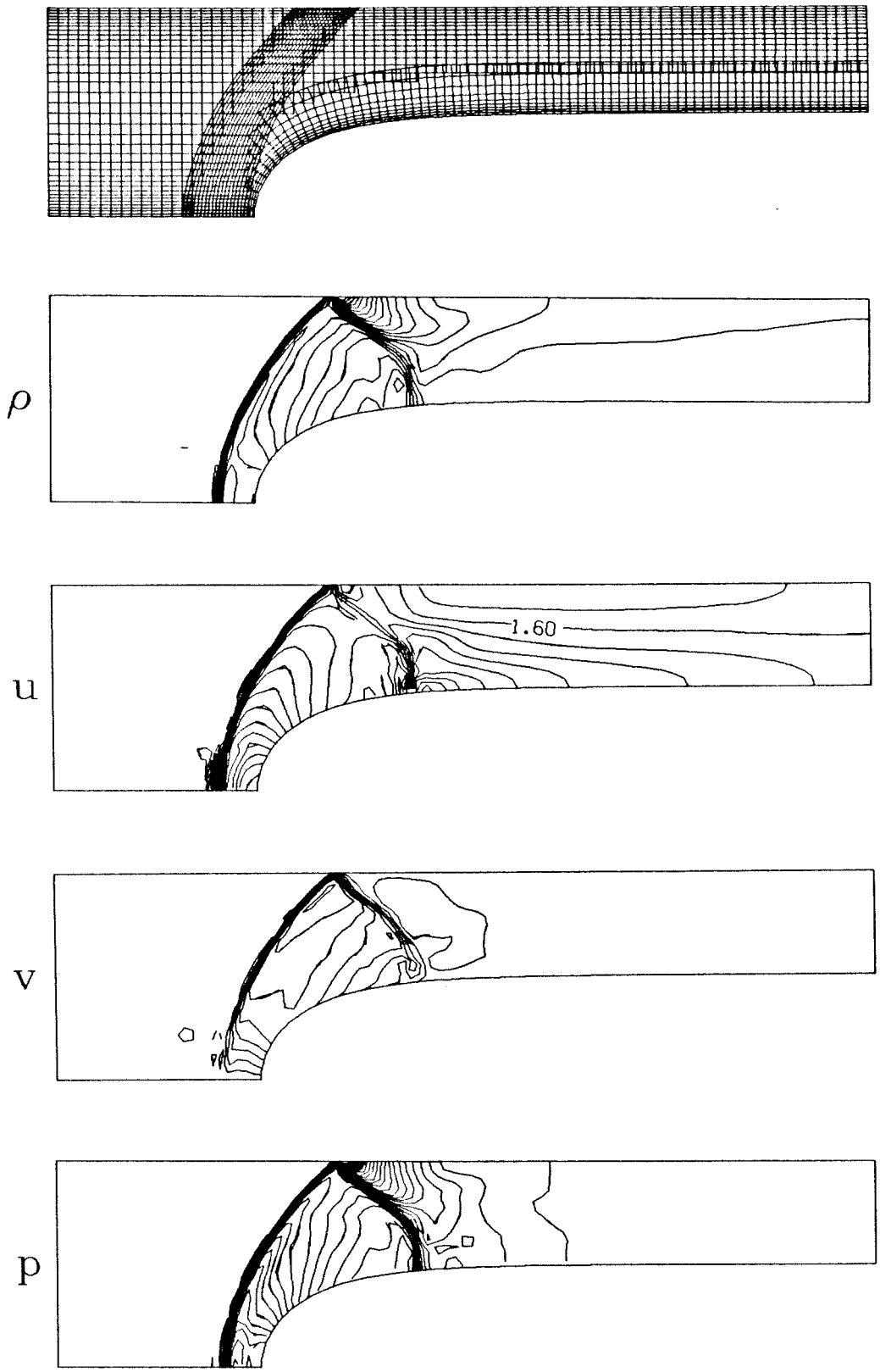


Fig. 4.4: Composite Grid and Steady State for the Second Step (with $\epsilon = 0.02$)

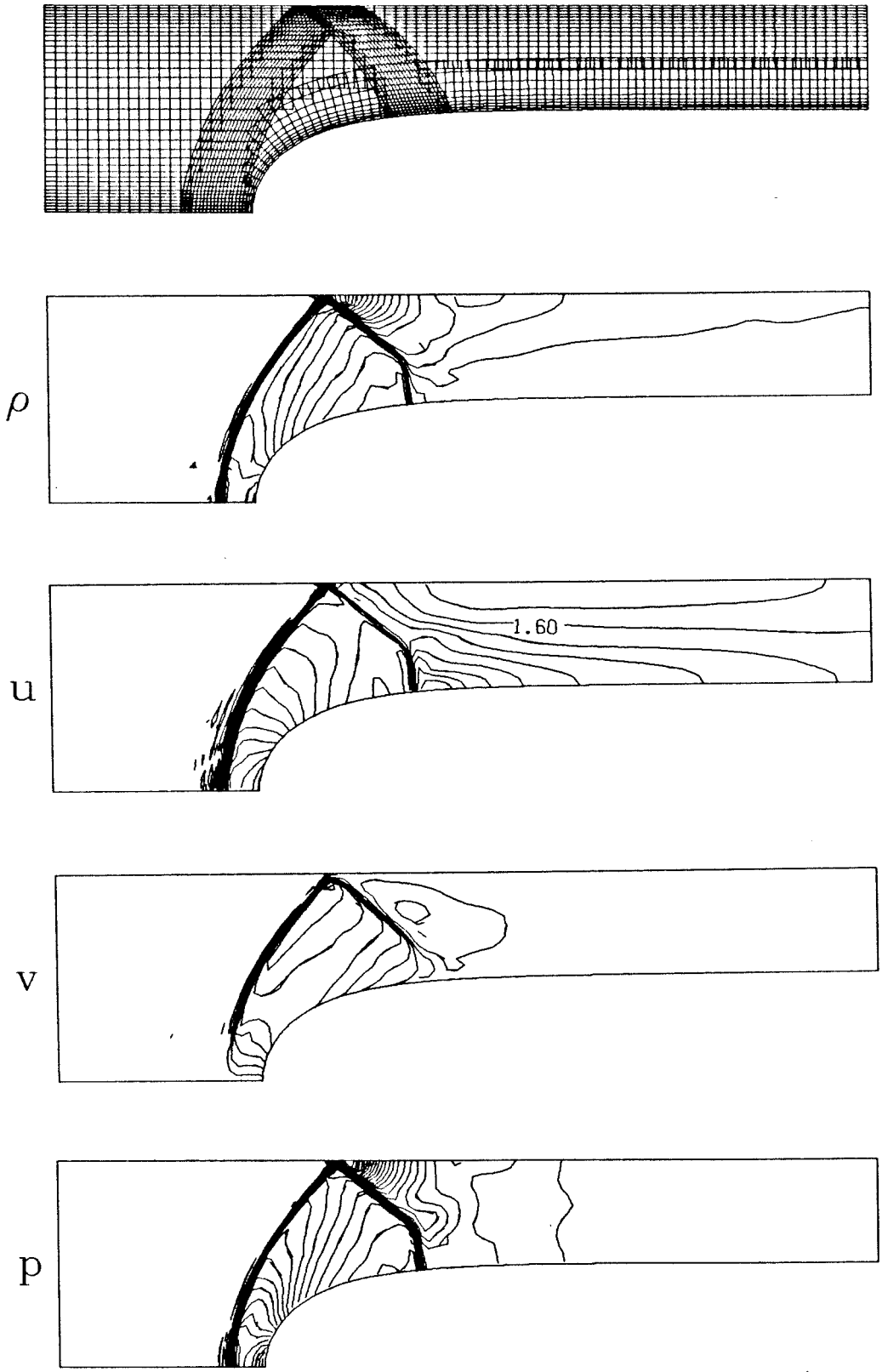


Fig. 4.5: Composite Grid and Steady State for the Third Step (with $\varepsilon = 0.015$)

APPENDIX 1

CMPGRD: A Composite Grid Construction Program*

A1.1 Purpose

For many problems that involve the discrete representation of a function on a two-dimensional region, a single rectangular grid is inappropriate. For example, a rectangular grid is poorly suited to the discretisation of a PDE boundary-value problem on any region other than a rectangle, because the boundary of the region will not in general correspond to a gridline. One alternative is to cover the region with several overlapping curvilinear grids, each of which is the image, under a smooth transformation, of a rectangular grid, and chosen so that the boundary of the region is composed entirely of edges of the curvilinear grids. Using enough of these transformations and grids it is possible to discretise problems on complicated regions for which a single grid is inadequate.

CMPGRD is an interactive FORTRAN program for generating composite grids in the plane, on a cylinder or on a torus. Any compact region with a smooth boundary can be covered with a composite grid, even a multiply-connected region. A composite grid for a region $\mathbf{D}$ in the plane is a set of smooth transformations from the unit square to regions in the plane which overlap to cover $\mathbf{D}$, and uniform grids on the unit square that are mapped by these transformations onto curvilinear grids in the plane. It provides boundary-fitted coordinates and allows stretched coordinates for boundary layers and internal layers wherever these are needed. The

* A User's Manual for program CMPGRD

above definition of a composite grid extends easily to regions with cylindrical or toroidal symmetry, and to n-dimensional regions. Program CMPGRD also could be generalised to generate n-dimensional composite grids, but at present it is restricted to two-dimensional regions that can be represented in the plane.

A1.2 Elements of a Composite Grid

The essential elements of a composite grid for a compact region $\mathbf{D}$ in the plane, on a cylinder or on a torus are a set of smooth transformations $\mathbf{d}_k: [0,1]^2 \mapsto \mathbb{R}^2$ chosen such that the union of the images $\mathbf{D}_k$ of the unit square under $\mathbf{d}_k$ contains the region $\mathbf{D}$ and the union of the boundaries of $\mathbf{D}_k$ contains the boundary of $\mathbf{D}$, and a set of uniform rectangular grids $\mathbf{G}_k$ on the unit square.

$$\mathbf{D} \subseteq \bigcup_k \mathbf{D}_k \quad \text{and} \quad \partial\mathbf{D} \subseteq \bigcup_k \partial\mathbf{D}_k, \quad \text{where}$$

$$\mathbf{D}_k = \{\mathbf{d}_k(r,s): 0 \le r \le 1, 0 \le s \le 1\}, \quad \text{and}$$

$$\mathbf{G}_k = \left\{ \left(\frac{i-1}{m_k - 1}, \frac{j-1}{n_k - 1} \right) : i = 1, \ldots, m_k,\ j = 1, \ldots, n_k \right\}.$$

The choice of transformations is free to the extent that there is more than one way to choose the $\mathbf{D}_k$ to cover $\mathbf{D}$ and there is more than one way to choose a smooth transformation $\mathbf{d}_k$ under which the unit square is mapped onto $\mathbf{D}_k$. The choice of $\mathbf{D}_k$ is restricted by the requirement that the transformations $\mathbf{d}_k$ must be smooth. Just how smooth the $\mathbf{d}_k$ need to be will depend on the specific application the grid is to be used for. For example, if the grid is to be used to discretise a second-order partial differential equation, the jacobian derivative of $\mathbf{d}_k$ must be continuous and nonsingular for $\mathbf{d}_k \in \mathbf{D}$. Anticipating this use, we shall consider only composite grids composed of transformations $\mathbf{d}_k$ with continuous, nonsingular jacobians. A consequence of this restriction is that the boundary of each $\mathbf{D}_k$ must consist of four smooth sides and four corners with interior angles smaller than π, but even with this restriction there is considerable freedom in the choice of the regions $\mathbf{D}_k$. The

boundary of $\mathbf{D}$ consists of smooth closed curves; the boundary of $\mathbf{D}_k$ consists of segments of these and other smooth curves. The following sections describe how these curves may be specified, how $\mathbf{D}_k$ may be specified in terms of these curves, and how the transformations $\mathbf{d}_k$ may be specified in terms of the boundary of $\mathbf{D}_k$. For the sake of clarity we will first consider only regions $\mathbf{D}$ in the plane. The generalisation to regions on the cylinder or torus is then straightforward.

A1.2.1 First A Simple Example

Most of the concepts involved in constructing a composite grid for any two-dimensional region are illustrated by the simple case of the unit disc in the plane,

$$\mathbf{D} = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \leq 1\}.$$

For this region we can construct, for example, a composite grid consisting of two transformations

$$\mathbf{d}_1(r, s) = ((0.4 + 0.6r)\cos 2\pi s, (0.4 + 0.6r)\sin 2\pi s))$$

$$\mathbf{d}_2(r, s) = (-0.6 + 1.2r, -0.6 + 1.2s)$$

from the unit square $[0, 1]^2$ onto the regions

$$\mathbf{D}_1 = \{(x, y) : 0.4 \leq x^2 + y^2 \leq 1.0\}$$

$$\mathbf{D}_2 = \{(x, y) : -0.6 \leq x \leq 0.6, -0.6 \leq y \leq 0.6\}$$

as shown in Figure A1.1, and uniform grids

$$\mathbf{G}_1 = \left\{ \left(\frac{i-1}{6}, \frac{j-1}{32} \right) : i = 1, \ldots, 7,\ j = 1, \ldots, 33 \right\}$$

$$\mathbf{G}_2 = \left\{ \left(\frac{i-1}{12}, \frac{j-1}{12} \right) : i = 1, \ldots, 13,\ j = 1, \ldots, 13 \right\}$$

on the unit square, as shown in Figure A1.2. The two regions overlap, and their union is the region $\mathbf{D}$, so the two grids and transformations are a composite grid for $\mathbf{D}$. This composite grid can be constructed in three steps:

(1) Specify parametrisations on $[0,1]$ for the curves forming the boundary of the images, under the proposed transformations $\mathbf{d}_k$, of the unit square.

$$\mathbf{c}_1(\xi) = (\cos 2\pi\xi, \sin 2\pi\xi)$$
$$\mathbf{c}_2(\xi) = (0.4\cos 2\pi\xi, 0.4\sin 2\pi\xi)$$
$$\mathbf{c}_3(\xi) = (0.4 + 0.6\xi, 0)$$
$$\mathbf{c}_4(\xi) = (-0.6 + 1.2\xi, -0.6)$$
$$\mathbf{c}_5(\xi) = (-0.6 + 1.2\xi, +0.6)$$
$$\mathbf{c}_6(\xi) = (-0.6, -0.6 + 1.2\xi)$$
$$\mathbf{c}_7(\xi) = (+0.6, -0.6 + 1.2\xi)$$

(2) Use these curves to specify the transformations $\mathbf{d}_k$. One option is to use a simple "Coons patch" [7] transformation to specify

$$\begin{aligned}\mathbf{d}_1(r,s) &= (1-s)\mathbf{c}_3(r) + s\mathbf{c}_3(r) + (1-r)\mathbf{c}_2(s) + r\mathbf{c}_1(s) \\ &\quad - (1-s)(1-r)\mathbf{c}_3(0) - s(1-r)\mathbf{c}_3(0) + (1-s)r\mathbf{c}_3(1) - sr\mathbf{c}_3(1) \\ &= [(1-r)\mathbf{c}_2(s) + r\mathbf{c}_1(s)] + [\mathbf{c}_3(r) - (1-r)\mathbf{c}_3(0) - r\mathbf{c}_3(1)] \\ &= (1-r)\mathbf{c}_2(s) + r\mathbf{c}_1(s) = ((0.4+0.6r)\cos 2\pi s, (0.4+0.6r)\sin 2\pi s)\end{aligned}$$

$$\begin{aligned}\mathbf{d}_2(r,s) &= (1-s)\mathbf{c}_4(r) + s\mathbf{c}_5(r) + (1-r)\mathbf{c}_6(s) + r\mathbf{c}_7(s) \\ &\quad - (1-s)(1-r)\mathbf{c}_4(0) - s(1-r)\mathbf{c}_5(0) + (1-s)r\mathbf{c}_4(1) - sr\mathbf{c}_5(1) \\ &= [(1-r)\mathbf{c}_6(s) + r\mathbf{c}_7(s)] + (1-s)[\mathbf{c}_4(r) - (1-r)\mathbf{c}_4(0) - r\mathbf{c}_4(1)] \\ &\quad + s[\mathbf{c}_5(r) - (1-r)\mathbf{c}_5(0) - r\mathbf{c}_5(1)] \\ &= (1-r)\mathbf{c}_6(s) + r\mathbf{c}_7(s) = (-0.6 + 1.2r, -0.6 + 1.2s)\end{aligned}$$

(3) Specify the grids $\mathbf{G}_1$ and $\mathbf{G}_2$ on the unit square by giving the number of gridlines in each direction on each grid:

$$m_1 = 7, \quad n_1 = 33, \quad m_2 = 13, \quad n_2 = 13.$$

A1.2.2 Regions in the Plane

Consider a compact region $\mathbf{D}$ in the plane, with a smooth boundary. In order to describe the boundaries of regions $\mathbf{D}_k$ whose union contains $\mathbf{D}$ and the union of whose boundaries contains the boundary of $\mathbf{D}$, we must define three categories of curves:

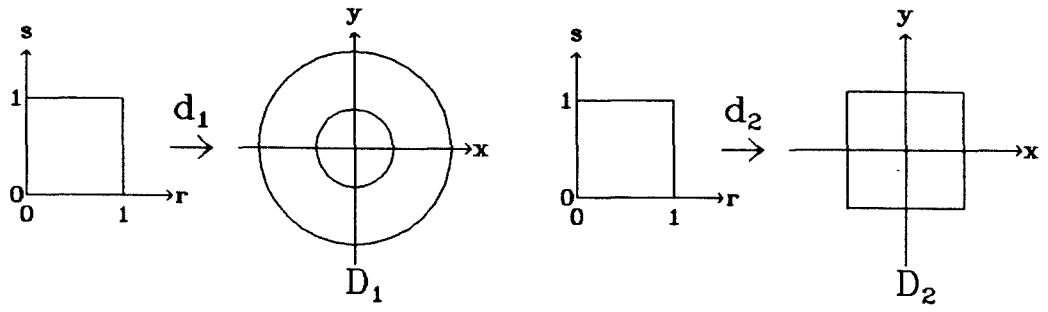


Figure A1.1: The unit square is mapped by $\mathbf{d}_1$ and $\mathbf{d}_2$ onto $\mathbf{D}_1$ and $\mathbf{D}_2$.

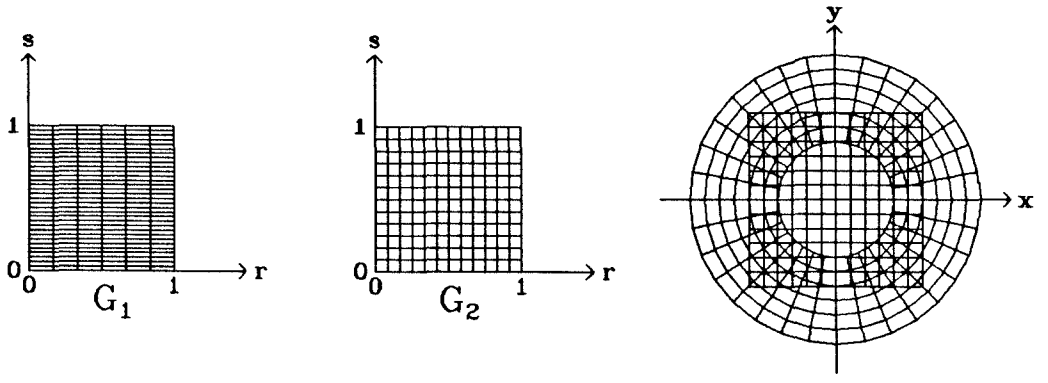


Figure A1.2: Grids $\mathbf{G}_1$ and $\mathbf{G}_2$ are mapped onto curvilinear grids.

(1) The closed curves $\mathbf{C}_k^B$ which form the boundary of $\mathbf{D}$. The boundary of a simply-connected region consists of exactly one such curve; that of a doubly-connected region consists of two, and so forth. These curves will be referred to as boundary curves.

(2) Curves $\mathbf{C}_k^P$ of which a segment is the image of two opposite sides of the unit square under a periodic transformation $\mathbf{d}_k$. Every $\mathbf{d}_k$ which maps one side of the unit square onto a segment of one of these curves must be periodic and map the opposite side of the unit square onto the same segment of that curve. These curves will be referred to as branch cuts.

(3) Other curves $\mathbf{C}_k^I$. No segment of one of these curves may lie on the boundary of $\mathbf{D}$ or be the image of two opposite sides of the unit square under any transformation $\mathbf{d}_k$.

Every curve used in constructing a composite grid falls into one of these categories.

In order to specify the transformations $\mathbf{d}_k$ in terms of segments of these curves, we must parametrise the curves. That is, we must specify continuous functions $\mathbf{c}_k: [0,1] \mapsto \mathbb{R}^2$ such that

$$\mathbf{C}_k = \{\mathbf{c}_k(\xi): 0 \leq \xi \leq 1\}.$$

For the example of the unit disc we defined parametrisations for one boundary curve $\mathbf{C}_1^B$, one branch cut $\mathbf{C}_3^P$, and five other curves $\mathbf{C}_2^I$, $\mathbf{C}_4^I$, $\mathbf{C}_5^I$, $\mathbf{C}_6^I$ and $\mathbf{C}_7^I$. The following three restrictions on the parametrisations $\mathbf{c}_k$ are neccessary but not sufficient to ensure that the transformations $\mathbf{d}_k$ may be smooth and have nonsingular jacobian derivatives:

(1) The functions $\mathbf{c}_k$ must have continuous, non-vanishing derivatives.

(2) The parametrisation $\mathbf{c}_k$ of every closed curve $\mathbf{C}_k$ must be periodic. Without loss of generality, we may insist that the period be one.

(3) Every closed curve $\mathbf{C}_k$ must intersect a branch cut and the intersection must not be tangential. Without loss of generality, we may insist that such an intersection occur at $\mathbf{c}_k(0) = \mathbf{c}_k(1)$.

In the unit disc example $\mathbf{C}_1^B$ and $\mathbf{C}_2^I$ were closed curves. Their parametrisations $\mathbf{c}_1$ and $\mathbf{c}_2$ were periodic and they intersected curve $\mathbf{C}_3^P$ at $\mathbf{c}_1(0) = \mathbf{c}_1(1) = (1,0)$ and $\mathbf{c}_2(0) = \mathbf{c}_2(1) = (0.4,0)$, respectively, at right angles to curve $\mathbf{C}_3^P$.

When we have specified and parametrised enough curves so that every edge of every $\mathbf{D}_k$ is a segment of some curve, then we are ready to specify the transformations $\mathbf{d}_k$. To avoid problems with notation, suppose we have numbered the curves so that segments of curves $\mathbf{C}_1$ and $\mathbf{C}_2$ form two opposite sides of $\mathbf{D}_k$, and segments of $\mathbf{C}_3$ and $\mathbf{C}_4$ form the other two sides, as indicated in the diagram, so that these

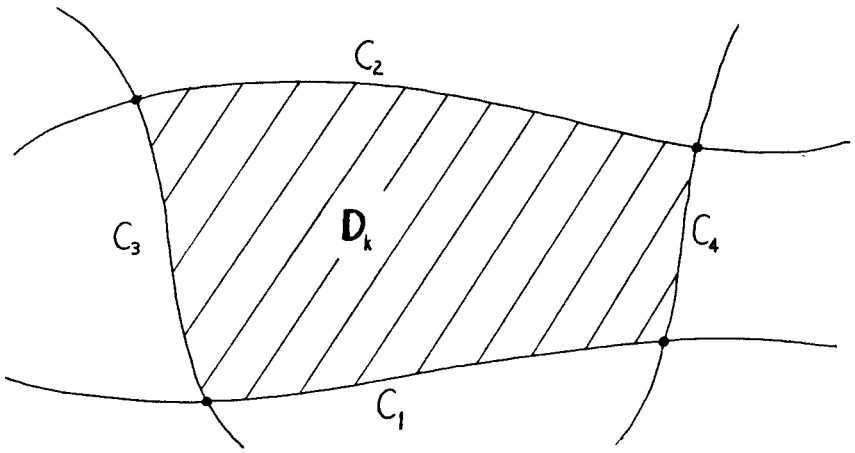


Figure A1.3: Curves Bounding a Region

curves intersect at

$$\mathbf{c}_1(\xi_{13}) = \mathbf{c}_3(\xi_{31}), \quad \mathbf{c}_1(\xi_{14}) = \mathbf{c}_4(\xi_{41}),$$

$$\mathbf{c}_2(\xi_{23}) = \mathbf{c}_3(\xi_{32}), \quad \mathbf{c}_2(\xi_{24}) = \mathbf{c}_4(\xi_{42}).$$

Then we must find a smooth transformation $\mathbf{d}_k$ such that

$$\mathbf{d}_k(r,0) = \mathbf{c}_1(\xi_{13} + (\xi_{14} - \xi_{13})r), \quad \mathbf{d}_k(r,1) = \mathbf{c}_2(\xi_{23} + (\xi_{24} - \xi_{23})r),$$

$$\mathbf{d}_k(0,s) = \mathbf{c}_3(\xi_{31} + (\xi_{32} - \xi_{31})r), \quad \mathbf{d}_k(1,s) = \mathbf{c}_4(\xi_{41} + (\xi_{42} - \xi_{41})r).$$

It may not be clear how best to choose such a transformation. The simplest option is a Coons patch, which is a linear combination of points on the edges of $\mathbf{D}_k$, in particular

$$\begin{aligned}\mathbf{d}_k(r,s) = {} & (1-s)\mathbf{d}_k(r,0) + s\mathbf{d}_k(r,1) + (1-r)\mathbf{d}_k(0,s) + r\mathbf{d}_k(1,s) \\ & - (1-s)(1-r)\mathbf{d}_k(0,0) - s(1-r)\mathbf{d}_k(0,1) \\ & - (1-s)r\mathbf{d}_k(1,0) - sr\mathbf{d}_k(1,1)\end{aligned}$$

A Coons patch will be adequate if the boundary of $\mathbf{D}_k$ is not too convoluted and the parametrisations of the curves are appropriate; otherwise, the transformation may contain folds. This problem can always be avoided by a careful choice of enough regions $\mathbf{D}_k$ so that the shape of each is simple enough that a Coons patch is adequate.

Another option is a conformal map. A conformal map from the unit square onto $\mathbf{D}_k$ can always be found, but it is not generally possible to ensure that the corners of the unit square correspond to the corners of $\mathbf{D}_k$ under such a transformation. Even when this is possible, the transformation will have a singularity at any corner of $\mathbf{D}_k$ that does not form a right angle. Related but more versatile methods exist [23], [21], [4] to determine a smooth transformation as the solution to a system of elliptic equations. However, if the boundary of $\mathbf{D}_k$ is too convoluted, these methods will result in a highly non-uniform transformation. As a result, these methods have in practice at most limited advantage over the less sophisticated Coons patch. For the example of the unit disc we used Coons patches for both $\mathbf{d}_1$ and $\mathbf{d}_2$, but conformal maps would have given exactly the same transformations in each case.

A1.2.3 Regions on a Cylinder or Torus

In case $\mathbf{D}$ lies on a cylinder or torus, we represent these as a strip and a box, respectively, in the plane. The period strip of the cylinder is specified by a pair of curves $(\mathbf{C}_1^P, \mathbf{C}_2^P)$ parametrised by two functions $\mathbf{c}_k$ which differ by a constant vector

$$\mathbf{c}_1(\xi) - \mathbf{c}_2(\xi) = \mathbf{c}_{12}$$

and have continuous, non-vanishing derivatives. The parametrisation should be chosen so that $\mathbf{c}_k(0)$ and $\mathbf{c}_k(1)$ lie on the boundary of $\mathbf{D}$. The period box of the torus is similarly specified by two pairs of curves $(\mathbf{C}_1^P, \mathbf{C}_2^P)$ and $(\mathbf{C}_3^P, \mathbf{C}_4^P)$, each pair parametrised by two functions $\mathbf{c}_k$ which differ by a constant vector

$$\mathbf{c}_1(\xi) - \mathbf{c}_2(\xi) = \mathbf{c}_{12}, \quad \mathbf{c}_3(\xi) - \mathbf{c}_4(\xi) = \mathbf{c}_{34}$$

and have continuous, non-vanishing derivatives. In this case, the derivative of $\mathbf{c}_k^P(\xi)$ should be periodic with period one

$$\frac{d\mathbf{c}_k}{d\xi}(\xi + 1) = \frac{d\mathbf{c}_k}{d\xi}(\xi), \quad k = 1, \ldots, 4$$

and the curves should intersect only at the four points

$$\mathbf{c}_1(0) = \mathbf{c}_3(0), \quad \mathbf{c}_1(1) = \mathbf{c}_4(0),$$

$$\mathbf{c}_2(0) = \mathbf{c}_3(1), \quad \mathbf{c}_2(1) = \mathbf{c}_4(1).$$

In most cases it is convenient for the curves specifying a period strip or period box to be straight lines. For example, if $\mathbf{D}$ is periodic in x with period a, bounded below by $y = 0$ and bounded above by $y = 1$ then we could specify the period strip by curves $(\mathbf{C}_1^P, \mathbf{C}_2^P)$ parametrised by

$$\mathbf{c}_1(\xi) = (0, \xi), \quad \mathbf{c}_2(\xi) = (a, \xi).$$

These curves do not at present fit into the category of branch cuts (those curves indicated by the superscript P) defined earlier. In order to accommodate them, that category is extended to include pairs of curves of which corresponding segments are images of opposite sides of the unit square under a transformation $\mathbf{d}_k$ whose derivative is periodic.

A1.3 PDEs on Composite Grids

The primary use of composite grids is the discretisation of partial differential equations. With appropriate difference methods they can be used to discretise elliptic, parabolic or hyperbolic PDE boundary-value problems. While a discussion of difference methods is beyond the scope of this work, the discretisation of a particular elliptic boundary-value problem is given as an example. Aside from that, the discussion is limited to how a PDE boundary-value problem may be expressed in cartesian coordinates on the unit square using the transformations of a composite grid.

A1.3.1 A Simple Example

Before looking at more general problems, it is instructive to consider the simple

example of Poisson's equation on the unit disc with Dirichlet boundary conditions:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y) \quad \text{on} \quad \mathbf{D} = \{(x,y): x^2 + y^2 < 1\}$$
$$u = g(x,y) \quad \text{on} \quad \partial\mathbf{D} = \{(x,y): x^2 + y^2 = 1\}.$$

This problem may be discretised using the composite grid discussed earlier, in particular the transformations

$$\mathbf{d}_1(r,s) = ((0.4 + 0.6r)\cos 2\pi s), (0.4 + 0.6r)\sin 2\pi s))$$

$$\mathbf{d}_2(r,s) = (-0.6 + 1.2r, -0.6 + 1.2s)$$

from the unit square onto the regions

$$\mathbf{D}_1 = \{(x,y): 0.4 \le x^2 + y^2 \le 1.0\}$$

$$\mathbf{D}_2 = \{(x,y): -0.6 \le x \le 0.6, -0.6 \le y \le 0.6\}$$

and grids

$$\mathbf{G}_1 = \left\{ \left(\frac{i-1}{6}, \frac{j-1}{32} \right) : i = 1, \ldots, 7,\ j = 1, \ldots, 33 \right\}$$
$$\mathbf{G}_2 = \left\{ \left(\frac{i-1}{12}, \frac{j-1}{12} \right) : i = 1, \ldots, 13,\ j = 1, \ldots, 13 \right\}$$

on the unit square. We represent the solution u by functions u_1 and u_2 on the unit square

$$u_1(r,s) = u(\mathbf{d}_1(r,s)), \quad u_2(r,s) = u(\mathbf{d}_2(r,s))$$

and write the boundary value problem in terms of u_1 and u_2. In the polar coordinates defined by $\mathbf{d}_1$, Poisson's equation becomes

$$\left(\frac{1}{0.6}\right)^2 \frac{\partial^2 u_1}{\partial r^2} + \left(\frac{1}{0.6(0.4 + 0.6r)}\right) \frac{\partial u_1}{\partial r} + \left(\frac{1}{2\pi(0.4 + 0.6r)}\right)^2 \frac{\partial^2 u_1}{\partial s^2} = f(\mathbf{d}_1(r,s))$$

and in the scaled coordinates defined by $\mathbf{d}_2$ it becomes

$$\left(\frac{1}{1.2}\right)^2 \left(\frac{\partial^2 u_2}{\partial r^2} + \frac{\partial^2 u_2}{\partial s^2} \right) = f(\mathbf{d}_2(r,s)).$$

The boundary condition becomes

$$u_1(1,s) = g(\mathbf{d}_1(1,s)).$$

In order to complete the specification of the problem in (r,s) coordinates and to make sure that u_1 and u_2 agree in the region of overlap, we apply matching conditions

$$u_1(r,s) = u_2\Big(\mathbf{d}_2^{-1}(\mathbf{d}_1(r,s))\Big)$$

on the inner edge of $\mathbf{D}_1$,

$$u_2(r,s) = u_1\Big(\mathbf{d}_1^{-1}(\mathbf{d}_2(r,s))\Big)$$

on all four sides of $\mathbf{D}_2$, and the periodicity condition

$$u_1(r,0) = u_1(r,1), \quad \frac{\partial u_1}{\partial s}(r,0) = \frac{\partial u_1}{\partial s}(r,1)$$

for $0 \leq r \leq 1$. A well-conditioned discretisation of the problem can be obtained using the grids $\mathbf{G}_1$ and $\mathbf{G}_2$ on the unit square. Using centered second-order differencing for the PDE and bilinear (second-order) interpolation for the matching conditions, a solution can be obtained which is second-order accurate in the limit that the number of gridlines in both directions on both grids tends to infinity while the two transformations remain fixed. This statement is misleading, since in practice as finer grids are considered, the transformations will be changed in such a way that the amount of overlap between grids decreases inversely as the number of gridlines. In this limit biquadratic (third-order) interpolation must be used to obtain a second-order accurate solution [12, §5.1].

A1.3.2 More General PDEs

Consider an arbitrary PDE boundary value problem

$$\mathbf{F}\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \mathbf{u}\right) = 0 \quad \text{for} \quad (x,y) \in \mathbf{D}$$

where $\mathbf{F}$ represents the PDE in the interior of $\mathbf{D}$ and the boundary conditions on the boundary of $\mathbf{D}$, and it is understood that higher derivatives may be involved.

Suppose we have a composite grid for $\mathbf{D}$ consisting of n transformations $\mathbf{d}_k$ from the unit square to regions

$$\mathbf{D}_k = \{\mathbf{d}_k(r,s): 0 \le r \le 1, 0 \le s \le 1\}.$$

For each k we can define a function $\mathbf{u}_k$ which represents $\mathbf{u}$ on the part of the unit square which is mapped by $\mathbf{d}_k$ onto $\mathbf{D}$,

$$\mathbf{u}_k(r,s) = \mathbf{u}(\mathbf{d}_k(r,s)) \quad \text{for} \quad \mathbf{d}_k(r,s) \in \mathbf{D}.$$

If the jacobian of the transformation $\mathbf{d}_k$ is non-singular we can use it to express the problem $\mathbf{F}$ in (r,s) coordinates. Let

$$J_k(r,s) = \frac{\partial \mathbf{d}_k}{\partial(r,s)} = \begin{pmatrix} x_r & x_s \\ y_r & y_s \end{pmatrix}.$$

Then the inverse of $J_k(r,s)$ is

$$J_k^{-1}(r,s) = \frac{1}{x_r y_s - y_r x_s} \begin{pmatrix} y_s & -x_s \\ -y_r & x_r \end{pmatrix} = \begin{pmatrix} r_x & r_y \\ s_x & s_y \end{pmatrix}.$$

With the understanding that by r_x, r_y, s_x and s_y we mean the functions of (r,s) which form the inverse of the jacobian, we can express the boundary value problem in (r,s) coordinates as

$$\mathbf{F}\left(r_x \frac{\partial}{\partial r} + s_x \frac{\partial}{\partial s}, r_y \frac{\partial}{\partial r} + s_y \frac{\partial}{\partial s}, \mathbf{u}_k\right) = 0 \quad \text{for} \quad \mathbf{d}_k(r,s) \in \mathbf{D}.$$

To specify the problem completely we will generally need to impose matching conditions in the regions of overlap between the $\mathbf{D}_k$. In order to do this in a systematic way it helps to be able to order the transformations $\mathbf{d}_k$ so that wherever two or more of the $\mathbf{D}_k$ overlap at the boundary of $\mathbf{D}$, this occurs at the boundary of the $\mathbf{D}_k$ with the highest index k of those involved. Although it is not always possible to order a given set of transformations in this way, it is always possible to choose an appropriate set of transformations that can be so ordered. To help in deciding on an appropriate sequence of transformations, imagine drawing $\mathbf{D}_1$ in the plane, next

drawing $\mathbf{D}_2$, erasing that part of $\mathbf{D}_1$ which overlaps $\mathbf{D}_2$, and so on, finally drawing $\mathbf{D}_n$ and erasing those parts of each underlying $\mathbf{D}_k$ which it overlaps. The result will be that the entire region $\mathbf{D}_n$ remains, along with those parts of each underlying $\mathbf{D}_i$ which are not covered by some $\mathbf{D}_j$ with $j > i$. If the sequence was chosen properly, the boundary of $\mathbf{D}$ is composed only of visible parts of the boundaries of the $\mathbf{D}_k$.

Now we are ready to specify the neccessary matching conditions. First we specify conditions identifying $\mathbf{u}_n$ with $\mathbf{u}_k$ on regions of overlap between $\mathbf{D}_n$ and the underlying regions $\mathbf{D}_k$. Where $\mathbf{D}_n$ overlaps other regions we can apply the matching condition

$$\mathbf{u}_n(r,s) = \mathbf{u}_k\Big(\mathbf{d}_k^{-1}(\mathbf{d}_n(r,s))\Big),$$

where k is the highest index of those regions which underlie $\mathbf{D}_n$ at $\mathbf{d}_n(r,s)$. The matching condition is applied on those sides of $\mathbf{D}_n$ which are specified in terms of curves $\mathbf{C}^I$ (other than boundary curves and branch cuts). Depending on the type of PDE involved, some higher derivatives of $\mathbf{u}_n$ and $\mathbf{u}_k$ may also need to be matched in the same way. In this way $\mathbf{D}_n$ is divided into two parts: the first part where the PDE and boundary conditions are applied and the second part a curve where the matching condition is applied. Matching conditions for the underlying regions are similarly specified. On region $\mathbf{D}_i$ we apply the condition

$$\mathbf{u}_i(r,s) = \mathbf{u}_j\Big(\mathbf{d}_j^{-1}(\mathbf{d}_i(r,s))\Big),$$

where j is the highest index of those regions which overlap or underlie $\mathbf{D}_i$ at $\mathbf{d}_i(r,s)$. The matching condition is applied on a curve (or curves) chosen so as to divide $\mathbf{D}_i$ into three parts: the first part where the PDE and boundary condition are applied, the second part a curve where the matching condition is applied, and a third part which includes any of $\mathbf{D}_i$ that is exterior to $\mathbf{D}$. This curve will lie along sides of $\mathbf{D}_i$ which are specified in terms of of curves $\mathbf{C}^I$ (other than boundary curves and branch cuts) wherever $i > j$, while for $i < j$ the curve lies on that part of $\mathbf{D}_j$ where the PDE and boundary condition are applied. By induction we can demonstrate

that it is possible to cover $\mathbf{D}$ with overlapping parts of all the $\mathbf{D}_k$ where the PDE is applied, each of which is separated from parts exterior to $\mathbf{D}$ by parts (curves) where a matching condition is applied. At this point the problem $\mathbf{F}$ is completely specified in (r, s).

A1.4 Using Program CMPGRD

CMPGRD is an interactive FORTRAN program for generating composite grids in the plane, on a cylinder or on a torus. It can generate a composite grid for any compact region with a smooth boundary, or even with a boundary that is only piecewise smooth (that is, a region with corners). The program and the instructions given in this manual both rely heavily on the use of an interactive graphics terminal, but it is possible with practice, patience and a good sketch of the region, to use the program without a graphics terminal or even in a batch mode. Specification of a composite grid is accomplished in three steps. First, curves which form the boundary of the region and of the individual grids are specified. Second, grids and transformations which form the composite grid are specified in terms of these curves. Third, the composite grid is specified in terms of these grids and transformations. Although composite grids are defined in terms of smooth transformations and the program also represents them internally as such, their primary use is in discretising PDE boundary value problems. For this purpose CMPGRD produces as output data a discrete description of the composite grid. This data may then be used by other programs to provide a finite-difference approximation to a PDE boundary value problem.

A1.4.1 Getting Started

Program CMPGRD is designed to be used interactively. At every step where it requires input it either gives instructions or supplies a menu of possible responses.

When the program starts, it gives the prompt

```
>
```

and waits for a response. If you type

```
HELP
```

then the program lists the menu

```
COMP   Specify a new composite grid.
CURVE  Specify a new curve.
EXIT   Exit after specifying one or more composite grids.
GRID   Specify a new grid.
HELP   Type this list.
PARAMS Specify new parameters.
QUIT
READ   Read command input from a file.
SAVE   Save command input in a file.
```

and repeats the > prompt. This menu lists all of the commands accepted after the > prompt. Any other response will result in an error message suggesting that you type HELP. The most important commands (other than HELP, of course) are COMP, CURVE and GRID. The prompt will change to COMP>, CURVE>, GRID> or PARAMS> when the corresponding command is entered; instructions are given after other commands whenever input is required. Use of the commands accepted after the > prompt is explained below.

COMP When one or more grids have been specified, this command is used to specify a composite grid for a region as a sequence of grids that overlap to cover the region. If the resulting composite grid is satisfactory then a discrete description of it may be saved in a data file in a form useful for discretising PDE boundary value problems. This command causes the prompt to change to COMP>, at which point a new menu of recognised commands is available. A later section of the manual is devoted to the use of this command.

CURVE Curves used as the sides of grids may be specified either by a subroutine supplied by the user or interactively using the cursor of a graphics terminal. This command is used to specify spline curves through data points entered by the graphics cursor. This command causes the prompt to change to CURVE>, at which point a new menu of recognised commands is available. A later section of the manual is devoted to the use of this command.

EXIT This command is the normal way to finish a session with CMPGRD. If no composite grid data has been saved in a file yet an error message to that effect is typed and the > prompt returns.

GRID When several curves have been specified, this command is used to specify a transformation from the unit square onto a region bounded by segments of these curves and a uniform grid on the unit square that corresponds to a curvilinear grid on this region. This command causes the prompt to change to GRID>, at which point a new menu of recognised commands is available. A later section of the manual is devoted to the use of this command.

HELP This command types a menu of recognised commands with a brief explanation of each.

PARAMS This command is used to set the size of the plotting area and parameters which determine the action taken by CMPGRD when it detects errors. This command causes the prompt to change to PARAMS>, at which point a new menu of recognised commands is available. A later section of the manual is devoted to the use of this command.

QUIT This command causes a session with CMPGRD to end immediately.

READ This command is used to read command input from a file saved with the SAVE command in a previous run of CMPGRD.

SAVE This command is used to save command input in a file for use in later runs of the program. The resulting file may be edited to remove erroneous input

or to insert other command input. The command input files saved from several runs may be combined and used as input to a subsequent run. In this way the individual curves and grids used to specify a composite grid may each be specified in separate runs of the program, followed by a run that specifies a composite grid.

A1.4.2 Specifying Curves

Curves are divided into three categories according to how they are used in specifying the grids and transformations of a composite grid. Some curves form parts of the boundary of the region to be covered by the composite grid. These are called boundary curves. Segments of some curves or pairs of curves are used as opposite sides of grids on which the problem is periodic. These are called branch cuts. Segments of all other curves are used only as sides of grids where they overlap other grids. Each side of a grid must be a segment of a curve that belongs to only one of these categories. Adjacent sides of a grid must be segments of different curves, so the endpoints of these segments are specified as intersections of curves. To ensure that these endpoints are well-defined and to avoid the possibility of highly-skewed grids, these intersections may not be tangential.

CMPGRD provides two methods by which curves may be specified. One is to to specify a cubic spline curve passing through data points entered using the cursor of a graphics terminal or from a data file. The other is to use a subroutine (supplied by the user) which specifies parametrisations for one or more curves. Spline curves have the advantage that they are easy to use in modelling regions whose boundaries cannot be described in terms of known functions. Curves specified by subroutine have the advantage that their parametrisations may be arbitrarily smooth functions, while cubic splines have discontinuous third derivatives. Grids may be specified in terms of curves specified by either method, and segments of curves specified by

different methods may be used as different sides of the same grid. Curves used as opposite sides of the period strip of the cylinder or the period box of the torus must occur in pairs. Spline curves may not be used for this purpose since they are specified only one at a time.

Subroutine CURVE must be supplied by the user. If the parametrisations of any curves are to be specified by subroutine, then CURVE gives these parametrisations. CURVE also gives the number of such curves, so even if there are none subroutine CURVE must still be supplied. At the beginning of a session, CMPGRD calls subroutine CURVE to find how many curves it specifies. Then it checks that the parametrisation of each curve has a non-vanishing derivative and that the parametrisations of pairs of branch cuts have identical derivatives. General specifications for subroutine CURVE and two examples are given in an appendix.

The CURVE command in the main menu (after the > prompt) is used to specify curves interactively. This command changes the prompt to CURVE>, at which point the curves currently specified are plotted for reference and the following menu of commands is available:

```
EXIT   Exit after specifying a curve.
HELP   Type this list.
QUIT
SPLINE Specify a spline curve.
```

This menu lists all of the commands accepted after the CURVE> prompt. The use of these commands is explained below.

EXIT This command is the normal way to return to the main menu and the > prompt after specifying a curve. If no new curve has been specified yet, an error message is typed and the CURVE> prompt returns.

HELP This command types a menu of recognised commands with a brief explanation of each.

QUIT This command abandons the curve currently being specified and returns to

the main menu and the > prompt.

SPLINE This command is used to specify a spline curve through data points entered interactively using the cursor of a graphics terminal. A spline curve is specified in three steps. First, CMPGRD asks whether the curve is a boundary curve, a branch cut or a curve used only for interpolated sides of grids. Second, CMPGRD asks if the problem is periodic on the curve. In particular, the problem is periodic on any curve which joins corresponding points on a pair of branch cuts. To help avoid the mistake of trying to specify a grid two opposite sides of which are incompatible segments of a pair of branch cuts, CMPGRD will not allow a non-periodic curve to begin or end on a branch cut. To help avoid the mistake of trying to specify a grid one side of which is a segment of a closed curve which is not parametrised continuously, CMPGRD will not allow a closed curve to begin or end anywhere except on a branch cut. Third, CMPGRD asks for points through which the spline passes. Four or more points are needed to specify a closed curve; other curves need at least two. Each point chosen may be either the cursor point or the nearest point on another curve, with the restrictions noted above. After all the points on a periodic curve have been entered, CMPGRD checks that the endpoints are compatible; the first point lies on a branch cut, so CMPGRD moves the last point to the corresponding point on the opposite branch cut.

A1.4.3 Specifying Grids and Transformations

The curvilinear grids which make up a composite grid are the images under smooth transformations of uniform grids on the unit square. To specify a grid we must specify a uniform grid on the unit square in the (r, s) plane and a transformation from the (r, s) plane to the (x, y) plane. The uniform grid on the unit square is

specified by the number of gridlines in each direction. The transformation from the (r,s) coordinates to (x,y) coordinates is specified in two stages. First, to allow for stretched coordinates we specify a transformation from the unit (r,s) square onto the unit (t,u) square. This transformation will be called the stretching function. Second, we specify a transformation from the unit (t,u) square onto a region $\mathbf{D}_k$ in the (x,y) plane bounded by segments of curves. This transformation will be called the mapping function. The transformation from the unit (r,s) square onto $\mathbf{D}_k$ is the composition of the stretching and mapping functions. Unless otherwise specified, the stretching is taken to be the identity function and the transformation is just the mapping function. Although the transformation could be specified in a single step, these two steps make stretched coordinates for boundary layers and interior layers easier to tailor to the needs of a specific problem.

CMPGRD provides a choice of two types of stretching functions from the unit square in the (r,s) plane to the unit square in the (t,u) plane. One option is the identity function $(t,u) = (r,s)$. The other option is an exponential stretching function. In this case t is a function of r only and u is a function of s only, and both of these functions have the same form. The inverses of these functions are weighted sums of a linear function and one or more terms involving hyperbolic tangent functions. In particular, t is related to r by the function

$$r(t) = \frac{t + \sum_{i=1}^{n} [\sigma_i(t) - \sigma_i(0)]}{1 + \sum_{i=1}^{n} [\sigma_i(1) - \sigma_i(0)]}$$

where

$$\sigma_i(t) = a_i \tanh b_i(t - c_i)$$

if the problem is not periodic in r, and

$$\sigma_i(t) = a_i \tanh b_i(t - c_i) + \sum_{j=1}^{\infty} a_i [\tanh b_i(t - c_i - j) + \tanh b_i(t - c_i + j)]$$

if the problem is periodic in r. Each term $\sigma_i(t)$ has three free parameters: the weight a_i, the exponent b_i and the location c_i. Each term produces a stretching

in the t coordinate centered at $t = c_i$. That is, the image in the (t, u) plane of a uniform grid in the (r, s) plane is a nonuniform rectangular grid with a maximum in density of gridlines for t near each c_i. The density of gridlines in the t-direction at the point (t, u) is proportional to the derivative $r'(t)$, and may be adjusted by varying the parameters a_i, b_i and c_i in each term. In particular, a_i is approximately the ratio of the number of gridlines in the stretched region of the grid near c_i to the total number of gridlines in unstretched regions, while b_i is approximately the ratio of the density of gridlines near c_i to the density of gridlines in unstretched regions. If the problem is periodic in r, an infinite series of correction terms is needed to ensure that the derivative $t'(r)$ of the stretching is periodic. For example, if there is one term in the stretching ($n = 1$), and we choose $a_1 = 1$, $b_1 = 1$ and $c_1 = 0$, the t coordinate is stretched at $t = 0$, as shown in Figure A1.4; if the problem is periodic in r then the t coordinate is equally stretched at $t = 1$. Another sum of the same form expresses s in terms of u, and together these specify $(t, u) = (t(r), u(s))$, where $t(r)$ and $u(s)$ are the inverses of functions given by weighted sums of linear and hyperbolic tangent terms.

CMPGRD provides only one type of mapping function from the unit square in the (t, u) plane to the region $\mathbf{D}_k$ in the (x, y) plane, namely a Coons patch [7]. This mapping is a linear combination of points on four segments of curves which form the boundary $\mathbf{D}_k$. To avoid problems with notation, suppose we have numbered the curves so that segments of curves $\mathbf{C}_1$ and $\mathbf{C}_2$ form two opposite sides of the region $\mathbf{D}_k$, and segments of $\mathbf{C}_3$ and $\mathbf{C}_4$ form the other two sides, so that these curves intersect at

$$\mathbf{c}_1(\xi_{13}) = \mathbf{c}_3(\xi_{31}), \quad \mathbf{c}_1(\xi_{14}) = \mathbf{c}_4(\xi_{41}),$$
$$\mathbf{c}_2(\xi_{23}) = \mathbf{c}_3(\xi_{32}), \quad \mathbf{c}_2(\xi_{24}) = \mathbf{c}_4(\xi_{42}).$$

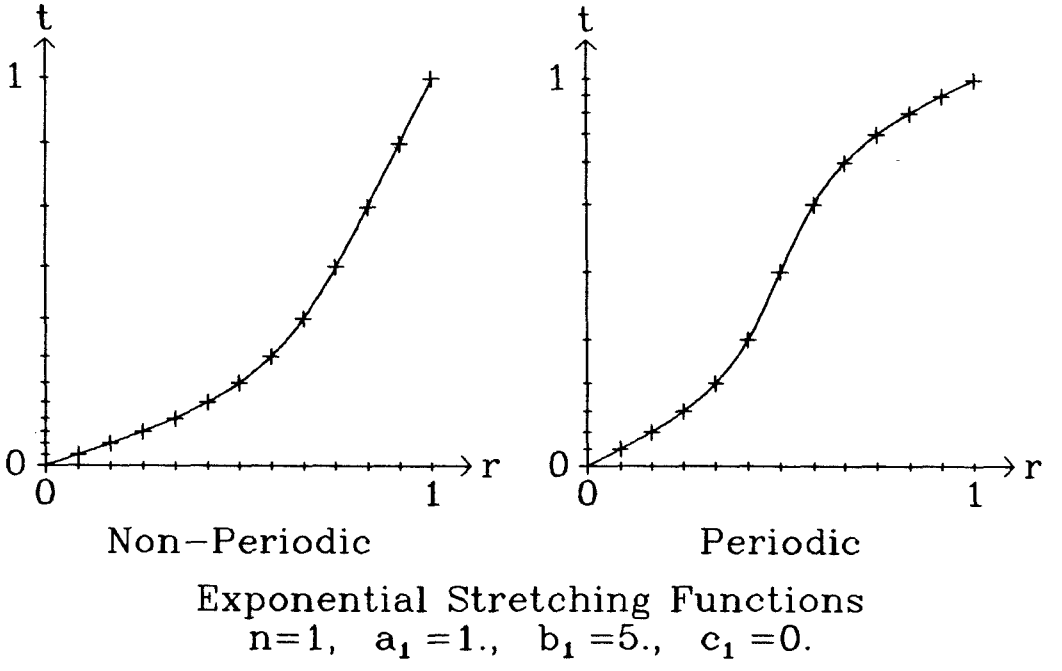


Figure A1.4: Examples of Stretching Functions

CMPGRD uses the Coons patch defined by

$$(x,y) = (1-u)\mathbf{c}_1(\xi_{13} + (\xi_{14} - \xi_{13})t) + u\mathbf{c}_2(\xi_{23} + (\xi_{24} - \xi_{23})t)$$
$$+ (1-t)\mathbf{c}_3(\xi_{31} + (\xi_{32} - \xi_{31})u) + t\mathbf{c}_4(\xi_{41} + (\xi_{42} - \xi_{41})u)$$
$$- (1-u)(1-t)\mathbf{c}_1(\xi_{13}) - u(1-t)\mathbf{c}_2(\xi_{23}) - (1-u)t\mathbf{c}_1(\xi_{14}) - ut\mathbf{c}_2(\xi_{24})$$

Since the parametrisations of the curves are smooth functions this mapping is also a smooth function. However, the constraint on the curve parametrisations that their derivatives not vanish does not imply that the jacobian derivative of the mapping is nonsingular or that the mapping is locally invertible. For example, if one of the sides of $\mathbf{D}_k$ has zero length then the jacobian will be singular on that side. In this case the resulting grid may still be useful for discretising a PDE boundary value problem if this singularity is taken into account. However, even if none of the sides has zero length the mapping may have folds, in which case it is not locally invertible and the resulting grid cannot be used for discretising a PDE. If this problem occurs,

one remedy is to subdivide the region $\mathbf{D}_k$ into two or more overlapping regions and try again. Despite this difficulty, the Coons patch mapping is still a useful tool.

The GRID command in the main menu (after the > prompt) is used to specify grids and transformations. This command causes the prompt to change to GRID>, at which point the curves currently specified are plotted for reference and the following menu of commands is available:

```
EXIT    Exit after specifying a grid.
HELP    Type this list.
LINES   Specify the number of gridlines in each direction.
MAPPING Select a mapping function (t,u) --> (x,y).
QUIT
SEGMENT Specify a segment of a curve.
SIDE    Associate a side of the grid with a segment.
STRETCH Specify a stretching function (r,s) --> (t,u).
```

This menu lists all of the commands accepted after the GRID> prompt. The prompt will change to MAPPING> or STRETCH> when the corresponding command is entered; instructions are given after other commands whenever input is required. Use of the commands accepted after the GRID> prompt is explained below.

EXIT This command is the normal way to return to the main menu and the > prompt after specifying a grid. If no new grid has been specified yet, an error message is typed and the GRID> prompt returns.

HELP This command types a menu of recognised commands with a brief explanation of each.

LINES This command is used to specify the numbers m_k and n_k of gridlines in the r and s directions respectively on the unit square.

MAPPING This command is used to select a mapping function and check that its jacobian is nowhere singular. At this time there is no choice of mapping function; a Coons Patch is always used. This command causes the prompt to change to MAPPING>, at which point the following menu of recognised commands is available:

```
COONS Use a Coons Patch mapping function.
EXIT  Exit after specifying a mapping function.
HELP  Type this list.
```

The COONS command is used to specify the Coons patch mapping function described above. If this mapping has a singular jacobian anywhere on the unit square then CMPGRD types a warning message to that effect.

QUIT This command abandons the grid currently being specified and returns to the main menu and the > prompt.

SEGMENT This command is used to specify a segment of a curve that forms one or more sides of the region in the (x, y) plane that the unit square is mapped onto. The curve and the endpoints of the segment on the curve are specified interactively using the cursor of a graphics terminal. First, the curve is chosen by entering a cursor point on or near it. The curve which passes nearest to the cursor point will be chosen. Second, the endpoints of the segment are chosen one at a time by entering cursor points at or near them. If the nearest curve to the cursor point is the curve already chosen then the endpoint will be the end of the curve nearest, in terms of its arcwise parameter, to a nearby point on the curve. Otherwise the endpoint will be a nearby intersection of the chosen curve and the curve nearest to the cursor point.

SIDE This command is used to specify which segment of a curve is associated with a side of the unit square in the (r, s) plane. CMPGRD displays the unit square and the segments that have so far been specified. First, a side of the unit square is chosen by entering a cursor point on or near it. Next, the associated segment is chosen by entering a cursor point on or near it. Finally, the orientation of the segment is reversed, if necessary, so that each endpoint of the side corresponds to the proper endpoint of the segment.

STRETCH This command is used to select a stretching function. The only available stretching function other than the identity is an exponential stretching. This command causes the prompt to change to STRETCH>, at which point the following menu of recognised commands is available:

```
EXIT  Exit after specifying a stretching function.
EXPO  Specify an exponential stretching function.
HELP  Type this list.
IDENT Use the identity stretching function.
```

The EXPO command is used to specify an exponential stretching of the type described above. CMPGRD first prompts for the direction (r or s) in which stretching is to be specified, then for the number of terms, and finally for the three coefficients a_i, b_i and c_i (discussed in §A1.4.3) for each term i. If the problem is periodic then the exponent in each term of the stretching in the periodic direction must be at least 1.0 so that the infinite series of correction terms converges quickly. This command must be repeated if stretching in both r and s is required.

A1.4.4 Specifying Composite Grids

A composite grid for a region $\mathbf{D}$ is a set of smooth transformations from the unit square onto regions which overlap to cover $\mathbf{D}$, and uniform grids on the unit square which are mapped by these transformations onto curvilinear grids. To be useful in discretising PDE boundary value problems, the grids and transformations comprising a composite grid must be given a hierarchy. That is, the composite grid is specified as a sequence of grids chosen so that wherever two or more grids overlap at the boundary of $\mathbf{D}$, the boundary corresponds to an edge of the grid that occurs latest in the sequence. Think of the grids as being laid down in sequence, so that the second grid obscures part of the first grid, the third grid obscures parts of the first and second grids, and so forth. All of the last grid in the sequence remains visible. If the sequence has been chosen correctly the boundary of $\mathbf{D}$ will consist

only of parts of visible edges of the grids.

The COMP command in the main menu (after the > prompt) is used to specify a composite grid and to save a discrete description of it in a data file in a form useful for discretising PDE boundary value problems. This command causes the prompt to change to COMP>, at which point the following menu of commands is available:

```
COORD Find coordinates of a point on composite grid.
EXIT  Exit after writing out data for composite grid.
GRIDS Specify the sequence of overlapping grids.
HELP  Type this list.
QUIT
SAVE  Save composite grid data in a file.
```

This menu lists all of the commands accepted after the COMP> prompt. The use of these commands is explained below.

COORD This command prompts prompts for cursor input of points (x, y) on $\mathbf{D}$ and returns (r, s), such that $(x, y) = \mathbf{d}_k(r, s)$ and k is the highest index of any grid from which the point can be interpolated. The results are also put into the log file.

EXIT This command is the normal way to return to the main menu and the > prompt after specifying a composite grid and saving data for it in a file. If data for a composite grid has not been saved yet, an error message is typed and the COMP> prompt returns.

GRIDS This command is used to specify a sequence of overlapping grids that make up a composite grid for a region $\mathbf{D}$. First, CMPGRD prompts for the number of grids in the composite grid. Next, it prompts for the sequence of grids, which must conform to the principle explained at the beginning of this section. Next, it prompts for the order of the PDE discretisation and of the interpolation. The order of the discretisation must be an even positive integer, and it determines the size of the computational molecule. For example, if the order of the discretisation is two then the computational

molecule will consist of nine points, while if the order of discretisation is four then the computational molecule will consist of 25 points. The order of the interpolation must be a positive integer, and it determines the type of interpolation and the number of points needed for interpolation. For example, if the order of the interpolation is two then bilinear interpolation can be used, which involves four points, while if the order of the interpolation is three then biquadratic interpolation will be used, which involves nine points. After the number of grids, their sequence and the order of the discretisation and of the interpolation is specified, CMPGRD determines which gridpoints will be used for the PDE and which for the matching conditions. If a valid sequence of grids was specified, all other gridpoints are either redundant or lie outside the region **D**. Otherwise, the regions of overlap do not separate all of the parts of the grids where a PDE could be discretised from parts that are exterior to the region **D**. CMPGRD plots those parts of each grid on which the PDE or the matching condition will be applied, leaving blank all unneeded parts. If the grids were specified in an invalid sequence or there was insufficient overlap between the grids then CMPGRD types an error message which reports that one or more of the grids will be null or that points needed for interpolation were deleted. In this case the composite grid cannot be used to discretise a PDE, and the sequence of grids must be respecified.

HELP This command types a menu of recognised commands with a brief explanation of each.

QUIT This command abandons the composite grid currently being specified and returns to the main menu and the > prompt.

SAVE This command is used to save data for the composite grid in a file for later use by programs that discretise PDE boundary value problems. The format

of the data in such a file is described in an appendix.

A1.4.5 Specifying Parameters

The PARAMS command in the main menu (after the > prompt) is used to set the size of the plotting area and parameters which determine the action taken by CMPGRD when it detects errors. This command causes the prompt to change to PARAMS>, at which point the following menu of recognised commands is available:

```
BOUNDS Specify new bounds for the plotting region.
DEBUG  Specify which items to save debug data for.
ERROR  Specify a new error handling procedure.
EXIT
HELP   Type this list.
```

This menu lists all of the commands accepted after the PARAMS> prompt. The use of these commands is explained below.

BOUNDS This command is used to specify new bounds for the plotting region. Initially, bounds of the plotting region are determined by extrema of the curves defined by subroutine curve, supplied by the user. These bounds may be inadequate if more curves are to be specified.

DEBUG This command is used for debugging. Internal variables used in generating a composite grid may be saved in a readable form in the log file. In addition, the common blocks may be saved when the program exits.

EXIT This command is used to return to the main menu and the > prompt.

HELP This command types a menu of recognised commands with a brief explanation of each.

ERROR This command is used to specify a new error handling procedure. Initially the procedure for handling errors is to crash the program (to get a traceback) only on fatal errors. Other errors (such as unrecognised commands) cause an error message to be typed and control to return to the terminal if a

command input file was being read at the time. Warnings are typed, but commands continue to be taken from a command input file if any was being read at the time. Informational messages are ignored. If the program is run as a batch job it is best to set the program to crash on all errors.

A1.5 Appendices

The following appendices give specifications and examples of subroutine CURVE, which must be supplied by the user; of the graphics package, which is specific to the graphics terminal and operating system; and of the composite grid data file which CMPGRD produces for later use by programs that discretise PDE boundary value problems.

A1.5.1 Subroutine CURVE

Subroutine CURVE must be supplied by the user. The details of the subroutine will vary with the geometry and topology of the region for which a composite grid is to be generated. General specifications and two examples are given here.

subroutine CURVE(k, s, x, y, xs, ys, kp)

Purpose:

Specify smooth curves $\mathbf{c}_k$ for $1 \leq k \leq kmax$, namely C^2 functions $\mathbf{c}_k$: $[0.0, 1.0] \mapsto \mathbb{R}^2$, their derivatives and information regarding their use.

Arguments on input:

k Type: Integer
Range: $0, 1, \ldots, kmax$

If $k = 0$, subroutine CURVE returns as kp the highest number, $kmax$, for which the subroutine defines a curve. The first call to subroutine CURVE is with $k = 0$, and all subsequent calls are with $1 \leq k \leq kmax$. If $k \geq 1$ then k is called the curve number.

s Type: Real
Range: $[0.0, 1.0]$ (see below)

s is the arc parameter of the curve, and should be in the range $[0.0, 1.0]$. However, sometimes subroutine CURVE is called with s outside this range, and although it may be a lot to ask, the definition of the curve should be extended to accommodate s at least in the range $[-0.1, 1.1]$. It is not essential that the derivative should be continuous on the extended interval. Just keep in mind that Newton's method is used extensively and is expected to work even when it involves calling subroutine CURVE near the endpoints of the interval $[0.0, 1.0]$.

Arguments on output:

x, y Type: Real

The value of the kth curve function at s. If $k < 1$ or $k > kmax$ then x and y need not be defined.

xs, ys Type: Real

The derivative of the kth curve function at s. If $k < 1$ or $k > kmax$ then xs and ys need not be defined.

kp Type: Integer

If $k = 0$, set $kp = kmax$, the highest number for which a curve is defined by subroutine CURVE. If $1 \leq k \leq kmax$ then kp indicates how curve k is to be used:

(a) If curve k is used as some part of the boundary of the region to be described, then $kp = -1$.

(b) If the composite grid is generated for a problem which is periodic on any of the component grids and curve k is used as a branch cut on one or two corresponding sides of such a grid, kp is the number of the curve which forms the branch cut on the opposite side. In this case the derivatives of curves k and kp must be equal for each s.

(c) If curve k is used only as an edge of a grid where that grid overlaps another grid, then $kp = 0$.

Two examples of subroutine CURVE are given below. The first specifies no curves at all. The second specifies seven curves, three of which may be used to form the boundary of an annular region, including a branch cut, and four of which bound a square.

```
1       subroutine curve(k,s,x,y,xs,ys,kp)
```

```
2 c       This subroutine specifies no curves at all.
3         kp=0
4         return
5        end


1       subroutine curve(k,s,x,y,xs,ys,kp)
2 c       The curves specified here can be used to specify a composite
3 c       grid for a disc of radius 1 consisting of two grids:
4 c       (1) an annular grid on the region between circles of radii 0.4
5 c           and 1.0 centered at the origin
6 c       (2) a rectangular grid on the square [-0.6,0.6]x[-0.6,0.6]
7         parameter(twopi=2.*3.14159265359)
8         if(k.eq.0)then
9 c         Seven curves are defined.
10          kp=7
11         elseif(k.eq.1)then
12 c        Curve 1 is the circle of radius 1.0 centered at the origin.
13          x=cos(twopi*s)
14          y=sin(twopi*s)
15          xs=-twopi*sin(twopi*s)
16          ys= twopi*cos(twopi*s)
17 c        Curve 1 forms the boundary of the region.
18          kp=-1
19         elseif(k.eq.2)then
20 c        Curve 2 is the circle of radius 0.4 centered at the origin.
21          x=0.4*cos(twopi*s)
22          y=0.4*sin(twopi*s)
23          xs=-twopi*0.4*sin(twopi*s)
24          ys= twopi*0.4*cos(twopi*s)
25 c        Curve 2 is the edge of the annular grid which overlaps the
26 c        rectangular grid.
27          kp=0
28         elseif(k.eq.3)then
29 c        Curve 3 joins curves 1 and 2 by a radial line.
30          x=0.4+0.6*s
31          y=0.
32          xs=0.6
33          ys=0.0
34 c        Curve 3 forms two opposite sides of the annular grid.
35          kp=3
36         elseif(k.eq.4)then
37 c        Curve 4 is the line y=-0.6 for -0.6.le.x.le.0.6
38          x=1.2*s-0.6
39          y=-0.6
40          xs=1.2
41          ys=0.0
42 c        Curve 4 is an edge of the rectangular grid which overlaps the
43 c        annular grid.
44          kp=0
45         elseif(k.eq.5)then
46 c        Curve 5 is the line y=0.6 for -0.6.le.x.le.0.6
47          x=1.2*s-0.6
48          y=0.6
49          xs=1.2
50          ys=0.0
51 c        Curve 5 is an edge of the rectangular grid which overlaps the
52 c        annular grid.
53          kp=0
54         elseif(k.eq.6)then
55 c        Curve 6 is the line x=-0.6 for -0.6.le.y.le.0.6
```

```
56            x=-0.6
57            y=1.2*s-0.6
58            xs=0.0
59            ys=1.2
60  c         Curve 6 is an edge of the rectangular grid which overlaps the
61  c         annular grid.
62            kp=0
63           elseif(k.eq.7)then
64  c         Curve 7 is the line x=0.6 for -0.6.le.y.le.0.6
65            x=0.6
66            y=1.2*s-0.6
67            xs=0.0
68            ys=1.2
69  c         Curve 7 is an edge of the rectangular grid which overlaps the
70  c         annular grid.
71            kp=0
72           endif
73          return
74         end
```

A1.5.2 The Graphics Package

All graphic input and output is handled by the following routines, which are specific to the graphics terminal and operating system and so must be written by the user. General specifications for these subroutines are given here, along with two examples of implementations of the graphics package.

subroutine SETUP(*ploter*)

Purpose:

Initialise graphics package and graphics terminal. This subroutine needs only to set the value of *ploter*. However, it may be used for any initialisation required by the graphics package and/or the graphics terminal.

Calling sequence:

This subroutine is called only once, before any other subroutine in the graphics package is called.

Arguments on output:

ploter Type: Integer

(a) If the terminal used does not have graphics capabilities then *ploter* = 0. In this case no other subroutines of the graphics package will be called.

(b) If the graphics terminal used is such that parts of a plot cannot be erased by plotting them over in the background colour, then *ploter* = 1. This is the case for most graphics terminals.

(c) If the graphics terminal used is such that parts of a plot can be erased by plotting them over in the background colour, then *ploter* = 2. This is case for some raster graphics terminals.

subroutine RESET

Purpose:

Reset the graphics package and terminal. This subroutine may be used for any resetting required by the graphics package and/or the graphics terminal.

Calling sequence:

This subroutine is called only once, after which no other subroutine in the graphics package is called.

subroutine STRTPL

Purpose:

Put the terminal into graphics mode.

Calling sequence:

This subroutine is called before each item is plotted. After the call to STRTPL and before the next call to ENDPLT, no alphanumeric output will be sent to the terminal by subroutine REPLY and no alphanumeric input will be expected by subroutine RQUEST of the input/output package.

subroutine ENDPLT

Purpose:

Take the terminal out of graphics mode.

Calling sequence:

This subroutine is called after each item is plotted. After the call to ENDPLT, alphanumeric output to the terminal from subroutine REPLY may be sent and alphanumeric input may be expected by subroutine RQUEST of the input/output package.

subroutine ERASE

Purpose:

Erase the graphics area of the terminal.

subroutine WINDOW(iw, xa, xb, ya, yb)

Purpose:

Set up a linear transformation onto the graphics area. CMPGRD often needs to draw two plots to be seen both at the same time. For this purpose the graphics area is conceptually divided up so that it contains two square regions called the primary and secondary windows. The primary window should take up most of the graphics area; the size of the secondary window is unimportant, but the two should not overlap. Subroutine WINDOW sets up a linear transformation from $[xa, xb] \times [ya, yb]$ onto one of two windows.

Arguments on input:

iw Type: Integer
Range: $1, 2$

If $iw = 1$ then use the primary (larger) window. If $iw = 2$ then use the secondary (smaller) window.

xa, xb Type: real
Range: $xa < xb$

xa and xb are coordinates of the left and right sides of the window under the linear transformation.

ya, yb Type: real
Range: $ya < yb$

ya and yb are coordinates of the bottom and top of the window under the linear transformation.

subroutine SHADE(sh)

Purpose:

Set colour, intensity or dash pattern for plotting. If the graphics terminal is capable of plotting in different colours, choose two different colours for $sh = 0.5$ and $sh = 1.0$, and use the background colour for $sh = 0.0$, which is used for erasing. If the graphics terminal is not capable of plotting in different colours or intensities then set up a dash pattern distinct from a solid line for use when $sh = 0.5$, set the dash pattern to solid for $sh = 1.0$ and to null for $sh = 0$. Bear in mind that items plotted with $sh = 0.5$ may be plotted over with $sh = 1.0$, and if $ploter = 2$, then items plotted with $sh = 1.0$ may be plotted over with $sh = 0.5$ or $sh = 0.0$ and items plotted with $sh = 0.5$ may be plotted over with $sh = 0.0$.

Arguments on input:

sh	Type:	real
	Range:	$\{0.0, 0.5, 1.0\}$

If $sh = 0.0$, set the colour to the background colour, or set the dash pattern to null. If $sh = 0.5$, set the colour to dim or gray, or set the dash pattern to something distinguishable from a solid line. If $sh = 1.0$, set the colour to bright or white, or set the dash pattern to solid.

subroutine MOVE(x, y)

Purpose:

Move the "pen" point to the image of (x, y) under the linear transformation set up by the most recent call to subroutine WINDOW. The point (x, y) should be in $[xa, xb] \times [ya, yb]$, but may be outside in some unimportant cases.

Arguments on input:

x, y	Type:	real
	Range:	$[xa, xb] \times [ya, yb]$ (see above)

The coordinates of the point.

subroutine DRAW(x, y)

Purpose:

Move the "pen" point to the image of (x, y) as in subroutine MOVE, and draw a line from the previous "pen" point.

Arguments on input:

x, y Type: real
Range: $[xa, xb] \times [ya, yb]$ (see above)

The coordinates of the point.

subroutine MARK(x, y)

Purpose:

Move the "pen" point to the image of (x, y) as in subroutine MOVE, and draw a small circle or other recognizable symbol there.

Arguments on input:

x, y Type: real
Range: $[xa, xb] \times [ya, yb]$ (see above)

The coordinates of the point.

subroutine CURSOR(x, y, key)

Purpose:

Obtain a cursor report from the graphics terminal using its joystick or equivalent, if any, and return the result (X,Y) under the transformation set up by the most recent call to subroutine window. Also return a one-character key entered from the keyboard. For example, on a Tektronix graphics terminal this can be the key that was hit in order to send the cursor report. This subroutine might be tricky to write since it may require that input from the terminal be read which is not terminated by a carriage-return. If this is too difficult or the graphics terminal has no joystick, then a simple way out is to enter (x, y) and the key directly from the keyboard using simple read statements.

Arguments on output:

x, y Type: real

The coordinates of the cursor under the transformation set up by the most recent call to subroutine window.

key Type: character*(*)

key(1:1) should be set to a one-character key entered from the the keyboard of the terminal.

Two examples of implementations of the graphics package are given below. The first is an implementation for use with no graphics terminal. This is not of much

use, since CMPGRD relies heavily on interactive graphics capabilities. Next is an implementation for use with a Tektronix 4107 graphics terminal. The final two listings contain system-dependent subroutines used by the latter implementation. They are for use on computers running the DEC VMS and BSD 4.2 Unix operating systems, respectively.

```
c
c     Graphics package for use with no graphics terminal.
c
      subroutine setup(ploter)
        integer ploter
        ploter=0
        return
       entry reset
       entry strtpl
       entry endplt
       entry erase
       entry window
       entry shade
       entry move
       entry draw
       entry mark
       entry cursor
       end
```

```
c
c     Graphics package for Tektronix 4107 terminal.
c
      subroutine setup(ploter)
        character buffer*512,str*3
        integer ploter,buflen
        logical alpha
        common/graph/alpha,xs,x0,ys,y0,buffer,buflen
        data alpha,buflen/.true.,0/
        call opterm
c       Select TEK code, clear dialog scroll and set dialog area to 10 lines.
        call encint(10,str,n)
        call bufout
     +    (char(27)//'%!0'//char(27)//'LZ'//char(27)//'LL'//str(:n))
        call erase
c       Set ploter=2 for raster graphic display.
        ploter=2
        return
       end

      subroutine reset
        character buffer*512,str*3
        integer buflen
        logical alpha
        common/graph/alpha,xs,x0,ys,y0,buffer,buflen
        call erase
c       Clear dialog scroll, set dialog area to 32 lines and select ANSI code.
        call encint(32,str,n)
        call bufout
```

```
30      +    (char(27)//'LZ'//char(27)//'LL'//str(:n)//char(27)//'%!1')
31           call wrterm(buffer,buflen)
32           call clterm
33           return
34          end
35
36          subroutine strtpl
37           return
38          end
39
40          subroutine endplt
41  c        Put the terminal back into alpha mode.
42           character buffer*512
43           integer buflen
44           logical alpha
45           common/graph/alpha,xs,x0,ys,y0,buffer,buflen
46           call bufout(char(31))
47           call wrterm(buffer,buflen)
48           alpha=.true.
49           return
50          end
51
52          subroutine erase
53  c        Erase the graphics area.
54           character buffer*512
55           integer buflen
56           logical alpha
57           common/graph/alpha,xs,x0,ys,y0,buffer,buflen
58           call bufout(char(27)//char(12))
59           call wrterm(buffer,buflen)
60           return
61          end
62
63          subroutine window(iw,xa,xb,ya,yb)
64  c        Set up a transformation from [xa,xb]x[ya,yb] onto a square.
65           character buffer*512
66           integer buflen
67           logical alpha
68           common/graph/alpha,xs,x0,ys,y0,buffer,buflen
69           if(iw.eq.1)then
70  c        Use the primary plotting region.
71             ia=256
72             ib=2303
73             ja=1024
74             jb=3071
75            else
76  c          Use the secondary plotting region.
77             ia=2816
78             ib=3839
79             ja=1536
80             jb=2559
81            endif
82           xs=(ib-ia)/(xb-xa)
83           x0=ia-xa*xs
84           ys=(jb-ja)/(yb-ya)
85           y0=ja-ya*ys
86           return
87          end
88
89          subroutine shade(sh)
90  c        Set the colour to black, light gray or white
```

```
      character*3 string
      if(sh.le.0.)then
        k=0
       elseif(sh.lt.1.)then
        k=15
       else
        k=1
       endif
      call encint(k,string,n)
      call bufout(char(27)//'ML'//string(:n))
      return
     end

     subroutine move(x,y)
c      Move the pen point to (x,y) with the pen up.
      character buffer*512,string*5
      integer buflen
      logical alpha
      common/graph/alpha,xs,x0,ys,y0,buffer,buflen
      call enccrd(x,y,string,n)
      call bufout(char(29)//string(:n))
      alpha=.false.
      return
     end

     subroutine draw(x,y)
c      Move the pen point to (x,y) with the pen down.
      character buffer*512,string*5
      integer buflen
      logical alpha
      common/graph/alpha,xs,x0,ys,y0,buffer,buflen
      if(alpha)then
        call bufout(char(29)//char(5))
        alpha=.false.
       endif
      call enccrd(x,y,string,n)
      call bufout(string(:n))
      return
     end

     subroutine mark(x,y)
c      Move the pen point to (x,y) and draw a small box there.
      character buffer*512,str1*3,str2*5
      integer buflen
      logical alpha
      common/graph/alpha,xs,x0,ys,y0,buffer,buflen
      call encint(8,str1,n1)
      call enccrd(x,y,str2,n2)
      call bufout
     +   (char(27)//'MM'//str1(:n1)//char(27)//'LH'//str2(:n2))
      return
     end

     subroutine cursor(x,y,key)
c      Get a cursor report.
      character key*(*),buffer*512,string*(8)
      integer buflen,hix,lox,sex,hiy,loy,fex
      logical alpha
      common/graph/alpha,xs,x0,ys,y0,buffer,buflen
c      Enable GIN.
      call bufout(char(27)//'IE01')
```

```
      call wrterm(buffer,buflen)
c     Read GIN report.
      call rdterm(string,len(string))
c     Reset bypass mode.
      call bufout(char(10))
      hiy=iand(ichar(string(2:2)),31)
      fex=ishft(iand(ichar(string(3:3)),12),-2)
      sex=iand(ichar(string(3:3)),3)
      loy=iand(ichar(string(4:4)),31)
      hix=iand(ichar(string(5:5)),31)
      lox=iand(ichar(string(6:6)),31)
      i=128*hix+4*lox+sex
      j=128*hiy+4*loy+fex
      x=(i-x0)/xs
      y=(j-y0)/ys
      key=string(1:1)
      return
     end

     subroutine encint(i,string,n)
c     Encode an integer.
      character string*(*)
      ii=iabs(i)
      i1=iand(ii,15)
      if(i.ge.0)i1=ior(i1,16)
      i2=iand(ishft(ii,-4),63)
      i3=iand(ishft(ii,-10),63)
      if(i3.ne.0)then
        n=3
        string=char(ior(i3,64))//
     +    char(ior(i2,64))//char(ior(i1,32))
       elseif(i2.ne.0)then
        n=2
        string=char(ior(i2,64))//char(ior(i1,32))
       else
        n=1
        string=char(ior(i1,32))
       endif
      return
     end

     subroutine enccrd(x,y,string,n)
c     Encode an (x,y) coordinate.
      character string*(*),buffer*512
      integer buflen,ox1,oy1
      logical alpha
      common/graph/alpha,xs,x0,ys,y0,buffer,buflen
      data ox1,oy1/2*-1/
      i=x*xs+x0
      j=y*ys+y0
      ny1=iand(ishft(j,-7),31)
      ny2=iand(ishft(j,-2),31)
      ny3=iand(      j     , 3)
      nx1=iand(ishft(i,-7),31)
      nx2=iand(ishft(i,-2),31)
      nx3=iand(      i     , 3)
      n=0
      if(ny1.ne.oy1)then
        n=n+1
        string(n:n)=char(ior(ny1,32))
       endif
```

```
213         n=n+1
214         string(n:n)=char(ior(ior(ishft(ny3,2),nx3),96))
215         n=n+1
216         string(n:n)=char(ior(ny2,96))
217         if(nx1.ne.ox1)then
218           n=n+1
219           string(n:n)=char(ior(nx1,32))
220          endif
221         n=n+1
222         string(n:n)=char(ior(nx2,64))
223         ox1=nx1
224         oy1=ny1
225         return
226        end
227
228       subroutine bufout(string)
229 c       Buffer a string to the terminal.
230         character string*(*),buffer*512
231         integer buflen
232         logical alpha
233         common/graph/alpha,xs,x0,ys,y0,buffer,buflen
234         do 10 i=1,len(string)
235           buflen=buflen+1
236           buffer(buflen:buflen)=string(i:i)
237           if(buflen.ge.512)call wrterm(buffer,buflen)
238 10       continue
239         return
240        end
```

```
  1       subroutine opterm
  2 c       Open the terminal for read and write.
  3         integer chan,flag,lib$get_ef,sys$assign
  4         common/termio/chan,flag
  5         if(.not.lib$get_ef(flag))stop'Error in OPTERM'
  6         if(.not.sys$assign('sys$output',chan,,))stop'Error in OPTERM'
  7         return
  8        end
  9
 10       subroutine clterm
 11 c       Close the terminal for read and write.
 12         integer chan,flag,lib$free_ef,sys$dassgn
 13         common/termio/chan,flag
 14         if(.not.lib$free_ef(flag))stop'Error in CLTERM'
 15         if(.not.sys$dassgn(%val(chan)))stop'Error in CLTERM'
 16         return
 17        end
 18
 19       subroutine rdterm(buffer,buflen)
 20 c       Purge terminal input and read the next buflen bytes into buffer
 21 c       with no echo and no interpretation of control characters.
 22         parameter(io$_readvblk=49,io$m_noecho=64,io$m_purge=2048)
 23         parameter(iocode=io$_readvblk.or.io$m_noecho.or.io$m_purge)
 24         character buffer*(*)
 25         integer buflen,chan,flag,trmblk(2),sys$qiow
 26         common/termio/chan,flag
 27         data trmblk/0,0/
 28         if(.not.sys$qiow(%val(flag),%val(chan),%val(iocode)
 29      +     ,,,,%ref(buffer),%val(buflen),,%ref(trmblk),,))
 30      +     stop'Error in RDTERM'
 31         return
```

```
      end

      subroutine wrterm(buffer,buflen)
c       Write buflen bytes to the terminal from buffer with no
c       interpretation of control characters; reset buflen to 0.
        parameter(io$_writevblk=48,io$m_noformat=256)
        parameter(iocode=io$_writevblk.or.io$m_noformat)
        character buffer*(*)
        integer buflen,chan,flag,sys$qiow
        common/termio/chan,flag
        if(buflen.gt.0)then
          if(.not.sys$qiow(%val(flag),%val(chan),%val(iocode)
     +       ,,,,%ref(buffer),%val(buflen),,,,))stop'Error in WRTERM'
          buflen=0
         endif
        return
       end
```

```
#include <sgtty.h>
      opterm_()
      /* Open the terminal for read and write. */
      {return;}
      clterm_()
      /* Close the terminal for read and write. */
      {return;}
      rdterm_(buffer,buflen)
      /* Flush terminal input and read the next
         buflen bytes into buffer with no echo. */
      char buffer[]; long *buflen;
      {
      int n; struct sgttyb state0,state1;
      if (ioctl(0,TIOCGETP,&state0) == -1) printf("error in rdterm");
      state1=state0;
      state1.sg_flags=state0.sg_flags+CBREAK-ECHO;
      if (ioctl(0,TIOCSETP,&state1) == -1) printf("error in rdterm");
      for ( n=0; n<*buflen; n++ )
        if (read(0,buffer+n,1) != 1) printf("error in rdterm");
      if (ioctl(0,TIOCSETP,&state0) == -1) printf("error in rdterm");
      return;
      }
      wrterm_(buffer,buflen)
      /* Write buflen bytes to the terminal
         from buffer; reset buflen to 0. */
      char buffer[];long *buflen;
      {
      if (*buflen == 0 ) return;
      if (write(1,buffer,(int)*buflen) != (int) buflen)
         printf("error in wrterm");
      *buflen = 0;
      return;
      }
        long iand_(i,j)
        /*  integer valued function to return logical AND of i and j */
        /*  i and j are assumed to be 4 byte integers  */
          long *i,*j;   {
                return(*i&*j);
        }
        long ior_(i,j)
        /*  integer valued function to return logical OR of i and j  */
        /*  i and j are assumed to be 4 byte integers  */
```

```
43          long *i,*j;   {
44               return(*i|*j);
45          }
46          long ishft_(i,nshift)
47          /*  shift the 4 byte integer i by nshift bits  */
48          /*  if nshift is negative, right shift end off zero fill  */
49          /*  if nshift is positive, left shift end around  */
50          /*  the routine behaves properly if magnitude of nshift > 32  */
51             long *i,*nshift;   {
52               long jshift,nbits;
53               if (*nshift<0)  {
54                  nbits = (*nshift < -32 ? 32 : -*nshift);
55                  jshift = (*i>>nbits) & (017777777777>>(nbits-1));
56               }
57               else  {
58                  nbits  = *nshift % 32;
59                  jshift = (*i<<nbits) | ( (*i>>(32-nbits))
60                                           & (~(037777777777<<nbits))  );
61               }
62               return(jshift);
63          }
```

A1.5.3 The Composite Grid Data File

CMPGRD produces an unformatted data file that contains all the information about a composite grid that is needed in order to discretise PDE boundary-value problems on it. This file may be read by a program written to discretise a PDE boundary-value problem. Its contents are explained below.

One record of four words: xa, xb, ya, yb.

> The rectangle $xa \leq x \leq xb$, $ya \leq y \leq yb$ bounds the region $\mathbf{D}$. This information is useful for plotting functions on the grid.

One record of one word: ng.

> ng is the number of component grids.

One record of $2ng$ words: $(m_k,\ n_k,$ for $k = 1, \ldots, ng)$.

> m_k and n_k are the number of gridlines in the r and s directions, respectively, on grid $\mathbf{G}_k$.

One record of $\sum_{k=1}^{ng} m_k n_k$ words: $(((kr_{ijk},$ for $i = 1, \ldots, m_k),$ for $j = 1, \ldots, n_k),$ for $k = 1, \ldots, ng)$.

> kr_{ijk} indicates whether the point (i, j) on grid $\mathbf{G}_k$ is a discretisation point, an interpolation point or neither. If $kr_{ijk} > 0$ then the point is a discretisation point; if $kr_{ijk} < 0$ then the point is an interpolation point;

if $kr_{ijk} = 0$ then the point is not used. The discretisation and interpolation points are numbered consecutively by $|kr_{ijk}|$ as first i, then j and then k increase. The exception to this is that corresponding points on corresponding branch cuts are given the same value of kr_{ijk}. In this way, when a PDE boundary-value problem is discretised on the composite grid, the discrete equation that applies at each discretisation and interpolation point is given a unique number, with the exception that the identical equations that apply at corresponding points on branch cuts share the same equation number.

One record of $5ng$ words: $(bc_k$, $(ibc_{ik}$, for $i = 1,\ldots,4)$, for $k = 1,\ldots,ng)$.

bc_k and ibc_{ik} indicate the boundary conditions on each side of each grid $\mathbf{G}_k$. bc_k is an integer which indicates the directions, if any, in which grid $\mathbf{G}_k$ is periodic. In particular, $bc_k = 0$ if grid $\mathbf{G}_k$ is not periodic; $bc_k = 1$ if grid $\mathbf{G}_k$ is periodic in the r direction only; $bc_k = 2$ if grid $\mathbf{G}_k$ is periodic in the s direction only; and $bc_k = 3$ if grid $\mathbf{G}_k$ is periodic in both directions. If $ibc_{ik} \neq 0$ then $|ibc_{ik}|$ is the number of the curve that side i of grid $\mathbf{G}_k$ is mapped to. $ibc_{ik} > 0$ indicates that side i corresponds to a boundary curve, $ibc_{ik} < 0$ indicates that side i corresponds to a branch cut, and $ibc_{ik} = 0$ indicates that side i is only interpolated from other grids.

One record of ng words: $(nb_k$, for $k = 1,\ldots,ng)$.

nb_k is the number of interpolated points on grid $\mathbf{G}_k$.

One record of $\sum_{k=1}^{ng} 3nb_k$ words: $((ib_{ik}$, jb_{ik}, kb_{ik}, for $i = 1,\ldots,nb_k)$, for $k = 1,\ldots,ng)$.

(ib_{ik}, jb_{ik}) form a list of the interpolated points on grid $\mathbf{G}_k$, and kb_{ik} indicates the grid from which each point is interpolated. Each interpolated point of grid $\mathbf{G}_k$ appears in the list as (ib_{ik}, jb_{ik}) for some i. The point (ib_{ik}, jb_{ik}) is interpolated from grid $\mathbf{G}_{kb_{ik}}$.

One record of $\sum_{k=1}^{ng} 2nb_k$ words: $((rb_{ik}$, sb_{ik}, for $i = 1,\ldots,nb_k)$, for $k = 1,\ldots,ng)$.

(rb_{ik}, sb_{ik}) are the coordinates of the interpolated point (ib_{ik}, jb_{ik}) on the grid $\mathbf{G}_{kb_{ik}}$ from which it is interpolated. The points used for interpolation and their interpolation weights can be computed from (rb_{ik}, sb_{ik}).

One record of $\sum_{k=1}^{ng} 2m_k n_k$ words: $(((x_{ijk}, y_{ijk}$, for $i = 1,\ldots,m_k)$, for $j = 1,\ldots,n_k)$, for $k = 1,\ldots,ng)$.

(x_{ijk}, y_{ijk}) are the coordinates of the point (i,j) on grid $\mathbf{G}_k$ under the transformation $\mathbf{d}_k$.

For $k = 1,\ldots,ng$, for $j = 1,\ldots,n_k$, for $i = 1,\ldots,m_k$, one record of four words: xr_{ijk}, yr_{ijk}, xs_{ijk}, ys_{ijk}.

xr_{ijk}, yr_{ijk}, xs_{ijk}, ys_{ijk} are the components of the Jacobian derivative

$\frac{\partial \mathbf{d}_k}{\partial(r,s)} = \begin{pmatrix} x_r & x_s \\ y_r & y_s \end{pmatrix}$ at the point (i,j) on grid $\mathbf{G}_k$.

APPENDIX 2

Multigrid on Composite Meshes*

W.D. Henshaw and G. Chesshire

Department of Applied Mathematics 217-50

California Institute of Technology

Pasadena CA 91125

Abstract. The multigrid method is applied to numerical solution of elliptic equations on general composite overlapping meshes. Computational results show that good convergence rates are obtained.

Key words. Composite meshes, overlapping grids, multigrid.

A2.1 Introduction

We describe the application of the multigrid method to the solution of elliptic partial differential equations (PDEs) on two dimensional regions which have been discretized using composite overlapping grids. A general purpose code, *CGMG*, has been developed (in FORTRAN) which can solve problems on composite meshes created by the grid construction program *CMPGRD*, Chesshire [6]. With *CMPGRD* the user may create a composite grid containing any number of component grids at any number of multigrid levels. CGMG can then be used to solve elliptic PDE boundary value problems.

* Submitted to SIAM J. Sci. Statist. Comput.

A composite overlapping grid consists of a number of simpler *component* grids. These component grids cover a region and overlap where they meet. Functions defined on the composite mesh are matched by interpolation at the overlapping grid boundaries. The problem of generating grids for regions of complicated geometry can be difficult, especially for those grid generation algorithms which attempt to fit a single global grid. With a composite overlapping grid, however, the component grids can be generated almost independently of each other. Each component grid can be stretched and refined with little effect on the other component grids. The numerical solution of PDEs on such grids has been examined by, among others, Starius [18], [19], Reyna [16], Kreiss [14], Atta [1], Benek et al. [2], Henshaw [12] and Berger [3].

The multigrid method is a fast iterative method for the solution of elliptic problems. Multigrid utilizes a sequence of grids of varying degrees of coarseness to accelerate the convergence of the solution on the finest grid. The basic principle rests on the fact that it is possible to obtain iterative procedures (*smoothers*) for which the high frequency components of the solution converge rapidly. This means that after a few smoothing iterations, the part of the solution yet to converge is smooth and hence can be accurately solved for on a coarser grid.

In the rest of this paper we outline the implementation of the multigrid algorithm on fairly general composite meshes. Multigrid on a model composite mesh is described in Stüben and Trottenberg [20] and, for more general two component meshes, in Henshaw [12]. Numerical results are presented for some test cases. These results show that the good convergence rates expected from multigrid can be obtained on composite meshes.

Acknowledgements. Computations were performed on the Fluid Dynamics Vax 11/750 at the California Institute of Technology. Support for this work came from the National Science Foundation under contract DMS-8312264 and the Office

of Naval Research under contract N00014-83-K-0422.

A2.2 Description.

The multigrid code *CGMG* was written to solve linear, variable coefficient elliptic PDEs of the form

$$(1a) \qquad Lu := c_{xx}\frac{\partial^2 u}{\partial x^2} + c_{xy}\frac{\partial^2 u}{\partial x \partial y} + c_{yy}\frac{\partial^2 u}{\partial y^2} + c_x\frac{\partial u}{\partial x} + c_y\frac{\partial u}{\partial y} + c_c u = f \quad \text{in } \Omega$$

$$(1b) \qquad Bu := b_x\frac{\partial u}{\partial x} + b_y\frac{\partial u}{\partial y} + b_c u = g \quad \text{on } \partial\Omega.$$

Periodic boundary conditions are also allowed. Currently a second order difference approximation has been implemented.

Let us first describe how the above PDE boundary value problem is discretized on a composite mesh, without any references to multigrid. The grid construction program CMPGRD can be used to generate a composite overlapping mesh for the region Ω. This mesh is composed of one or more component meshes. See, for example, the composite meshes of figures A2.1 and A2.2. Each component mesh is logically rectangular, although some points in overlap regions are discarded. A point on a component grid will be one of the following four types :

(i) An *interior* point where the PDE (1a) should be applied.

(ii) A *boundary* point, which corresponds to $\partial\Omega$, where the boundary condition (1b) holds.

(iii) A point where the solution is matched, by interpolation to the solution on other component grids. We sometimes call the set of all such interpolation points the *interpolation boundary.*

(iv) A point in a region of overlap which is not used.

There are a number of possible ways to generate the discrete approximations to the PDE (1a) the boundary conditions (1b) and the interpolation equations. The approach we take is a *mapping* method and proceeds as follows. Each component grid

is simple enough so that it can be mapped smoothly to a unit square (coordinates (r,s)). The PDE can be written in these (r,s) coordinates.

$$Lu = c_{rr}\frac{\partial^2 u}{\partial r^2} + c_{rs}\frac{\partial^2 u}{\partial r \partial s} + c_{ss}\frac{\partial^2 u}{\partial s^2} + c_r\frac{\partial u}{\partial r} + c_s\frac{\partial u}{\partial s} + c_c u = f$$

$$Bu = b_r\frac{\partial u}{\partial r} + b_s\frac{\partial u}{\partial s} + b_c u = g$$

The coefficients in these equations depend on derivatives of the mapping. These derivatives are supplied as output from the composite grid construction program CMPGRD. The solution of a PDE on a composite mesh can then be considered as the solution of PDE's on a sequence of unit squares. The solutions on the squares are coupled to each other through the interpolation boundaries. Let the discrete solution at point (i,j) on component grid k be denoted by $v_k(i,j)$. Then, for each k we obtain discrete approximations to the elliptic equation and the boundary conditions of the form

$$L_k v_k = f_k \tag{2a}$$

$$B_k v_k = g_k. \tag{2b}$$

A point (i,j,k) on an interpolation boundary is interpolated from some other component grid k'. CMPGRD supplies the position (r',s') of the point (i,j,k) on grid k'. Hence, standard interpolation formulae for rectangular grids can be applied :

$$v_k(i,j) = \sum_{i',j'} \alpha_{kk'}(i,j,i',j')v_{k'}(i',j'). \tag{2c}$$

These equations (2a), (2b) and (2c) will be written as a single linear system

$$\mathbf{A}\mathbf{v} = \mathbf{f}. \tag{3}$$

The vector $\mathbf{v}$ of all unknowns will be called a composite mesh function. $\mathbf{A}$ is an example of a composite mesh operator, mapping one composite mesh function to another composite mesh function.

The mesh equations (3) can be solved in any number of ways; using multigrid is but one possibility. If there are not too many equations, the system can be solved directly. Sparse matrix routines [11] have been used for this purpose. For large systems iterative methods become attractive. Many standard iterative methods are applicable. The matrix is not symmetric, however, so that some schemes do not apply. In general it is best to try and solve all the equations simultaneously, rather than iterating for too long on one component grid. In Stüben and Trottenberg [20] it was shown that an iteration based on the Schwartz alternating procedure (where the equations on each component grid are solved exactly before updating interpolation boundaries) is slower and sensitive to the amount of overlap. The general principle to follow seems to be to iterate in such a way that, at any time, all equations have converged to about the same degree.

A2.2.1 Multigrid.

The multigrid algorithm can be applied to the solution of the mesh equations (3). Discussions of the multigrid method in general can be found, for example, in Brandt [5], or Stüben and Trottenberg [20].

Once a composite mesh has been constructed using CMPGRD it is a simple matter to have CMPGRD generate the sequence of coarser composite meshes which are used for the multigrid algorithm. Figures A2.1 and A2.2 show some composite meshes which have been generated for multigrid. Denote the finest composite mesh, level 1, by $\mathbf{M}^1$ and successively coarser meshes by $\mathbf{M}^l, l = 2, 3, \ldots$. Note that for simplicity the composite meshes at the different levels all have the same number of component meshes. The elliptic PDE boundary value problem can be discretized on each of the composite meshes $\mathbf{M}^l$. Let $\mathbf{v}^l$ denote the composite mesh function for level l. Then at each level l there will be mesh equations of the form (3)

$$\mathbf{A}^l\mathbf{v}^l = \mathbf{f}^l. \tag{4}$$

Now let us outline the multigrid algorithm as it applies to composite meshes. Let $\mathbf{v}^1(p)$ be the pth iterate in the solution of the mesh equations on the finest mesh $\mathbf{M}^1$.

Multigrid Algorithm

while not converged **do**

smooth ν_1 times

$$\mathbf{v}^1(*) \leftarrow (\mathbf{S}^1)^{\nu_1}\mathbf{v}^1(p)$$

compute the defect and transfer to the coarser grid

$$\mathbf{f}^2 \leftarrow \mathbf{R}^{1\to 2}(\mathbf{f}^1 - \mathbf{A}^1\mathbf{v}^1(*))$$

"solve" the defect equation on the coarser grid

$$\mathbf{v}^2 \leftarrow (\mathbf{A}^2)^{-1}\mathbf{f}^2$$

correct the fine grid solution from coarse grid solution

$$\mathbf{v}^1(**) \leftarrow \mathbf{v}^1(*) + \mathbf{P}^{2\to 1}\mathbf{v}^2$$

smooth ν_2 times

$$\mathbf{v}^1(p+1) \leftarrow (\mathbf{S}^1)^{\nu_2}\mathbf{v}^1(**)$$

end while

This is the basic defect correction scheme. The smoothing operator $\mathbf{S}^1$, the restriction operator $\mathbf{R}^{1\to 2}$ and the prolongation operator $\mathbf{P}^{2\to 1}$ will be described in the context of composite grids. The defect equation need only be "solved" approximately. This approximate solution can be obtained by multigrid, in which case the algorithm becomes recursive. At the coarsest level the equations are usually solved directly.

Smoothers. The smoothing operator $\mathbf{S}$ for composite grids consists of smoothing each component grid and updating the interpolation boundaries. A component grid may be smoothed with any of the standard smoothers that exist. The program CGMG allows the user to choose from a number of possibilities including Gauss–

Seidel, Red-Black-Gauss-Seidel, Zebra line smoothers and alternating Zebra line smoothers. Each component grid can have a different smoother; the smoother can be tailored to the grid. For example, a particular component grid might be stretched to resolve a boundary layer, in which case one can use a line smoother in the appropriate direction.

There is some freedom as to the order of smoothing and interpolation. One possibility is to smooth all component grids before updating the interpolation boundaries :

$$\mathbf{S}^l = \begin{cases} \text{Smooth first component grid} \\ \text{Smooth second component grid} \\ \quad \vdots \\ \text{Smooth last component grid} \\ \text{Interpolate} \end{cases} .$$

From experience, however, it seems that a good procedure involves interpolating after each component grid is smoothed :

$$\mathbf{S}^l = \begin{cases} \text{Smooth first component grid} \\ \text{Interpolate} \\ \text{Smooth second component grid} \\ \text{Interpolate} \\ \quad \vdots \end{cases} .$$

The latter composite smoother requires more interpolations but this extra work is usually small compared to the smoothing operations.

Restriction Operators (Fine to Coarse Grid Transfer). The defect computed on a given level is transferred to the next coarsest level by the restriction operator $\mathbf{R}^{1\to 2}$. (The superscript $1 \to 2$ indicates that this operator maps mesh functions on $\mathbf{M}^1$ to mesh functions on $\mathbf{M}^2$.) A typical restriction operator determines the value at points on the coarse grid as some weighted average of the surrounding points on the fine grid. We use the so called full weighting restriction. The defects in the boundary equations are averaged separately from the defects in the interior equations; boundary defects are averaged along the boundary line. Since the final stage of the smoothing operation involves an interpolation, the de-

fects in the interpolation equations are all zero. Hence, no defect need be transfered at these points.

Prolongation Operators (Coarse to Fine Grid Transfer). The prolongation operator $\mathbf{P}^{2\to 1}$ maps the coarse grid solution to the fine grid. This mapping usually takes the form of an interpolation. We use second order interpolation. Interpolation boundary values could be corrected as well. However, once all other values have been corrected, the interpolation equations can be solved to update the interpolation boundary. It turns out that less overlap is needed on finer meshes for the latter approach.

Choice of Parameters and Cycle. An important part of the multigrid algorithm is the choice of the parameters ν_1, ν_2, etc. and the choice of cycle. *Cycle* is the term used to denote the sequence in which the different levels of grids are traversed. The program dynamically determines the type of cycle and the values for ν_1 and ν_2 in a manner similar to that described by Brandt [5]. There are two basic principles :

(i) Perform smoothing iterations until the smoothing rate (the reduction in residual per iteration) becomes larger than some value η, where $\eta \approx .6$. The smoothing typically starts out very fast but then slows down once the high frequencies have been reduced compared to the low frequencies.

(ii) Return to a finer grid once the residual at this level has been reduced by a certain factor δ with $\delta \approx .1$.

Ideally the program should automatically choose the types of component smoothers and the parameters η and δ. This could be done, for example, by examining component grids for stretching and by monitoring the smoothing rates. A program with this level of sophistication has not yet been developed. Currently one must make some educated guesses and then examine the results to determine any changes that might improve convergence rates.

Remarks on Grid Construction. For practical reasons the amount of overlap between component grids is kept fairly small. This reduces the number of computational points. With sufficient overlap, points on an interpolation boundary can be explicitly interpolated from non-interpolation points on other component grids. However, as the amount of overlap decreases the interpolation points may become coupled through the interpolation equations. In this case a system of equations must be solved to obtain values on the interpolation boundaries in terms of other values. Of course, as the overlap goes to zero these equations may become singular. The behaviour of the numerical solution to a model elliptic problem, as a function of the amount of overlap, was considered in Henshaw [12]. Suppose that the amount of overlap goes to zero as the grid is refined. To maintain accuracy, the order of accuracy of the interpolation formulae must be greater than the order of accuracy of the interior formulae. In the results presented here we use second order accurate approximations to the elliptic equations and require the amount of overlap to be greater than one half a grid line. In this case the interpolation formulae should be third order accurate.

To make the restriction and prologation operators simple, the composite meshes at different multigrid levels are strongly related. For example, interior points on a coarse grid coincide with interior points on finer grids. In Henshaw [12] the coarse grid was generated first and all finer grids were generated by simply doubling the number of grid lines. This gave reasonble results for composite grids with 2 component grids. However, this method does not immediately generalize to more component grids, and it tends to lead to more overlap on the finer grids than is really necessary. Hence, the grid construction program CMPGRD was designed explicitly to meet the requirements of the multigrid routine [6].

Remarks on Programming. A computer code for solving problems on composite meshes is somewhat more complicated than a code written for a single grid.

One of the major difficulties arises in the handling of all the data that descibe the composite grid. We have worked out a scheme for storing the composite grid data in an efficient and flexible manner. Physically, all the data is stored on a single array. Logically, the data are stored in a "directory-tree" fashion. Variables can be stored or accessed by name using utility routines. Arrays can be stored with a minimum of wasted space. With this storage structure, the composite grid data can be easily passed to subroutines.

A2.3 Numerical Results

In this section we present results from the multigrid solver CGMG. The composite grids used in the examples are shown in figures A2.1 and A2.2. Grid 1 (figure A2.1) has 3 levels and Grid 2 (figure A2.2) has 2 levels. The interpolation points are marked with small circles. In both examples we solve

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f \tag{4}$$

$$f = -2\pi^2 \cos \pi x \cos \pi y.$$

The boundary conditions are chosen so that the true solution is

$$u_{\text{true}} = \cos \pi x \cos \pi y.$$

Test 1 The Poisson equation (4) is solved on Grid 1. This grid might to be used to study flow around an obstacle in a varying channel. Periodic boundary conditions are applied at the left and right ends of the channel. Dirichlet boundary conditions, $u = u_{\text{true}}$, are given at all other boundaries. For smoothers we chose Red–Black on the rectangular grid and line–Zebra smoothers for the 3 curvilinear grids. The Zebra smoothers are on lines normal to the boundaries, since this is the direction in which the grids are stretched. Smoothing was performed until the smoothing rate became greater than $\eta = .6$. The residual at level 2 had to be decreased by a factor of $\delta = .01$ before the program would return to level 1. These parameters resulted in

	Grid 1		Grid 2	
Iteration	r(p)/r(p-1)	ECR	r(p)/r(p-1)	ECR
p=1	.019	.71	.091	.73
p=2	.065	.72	.033	.63
p=3	.055	.71	.11	.75
p=4	.385	.89	.14	.77

Table A2.1: Convergence Rates

a "W" cycle; the levels were traversed in the order (1,2,3,2,3,2,1). A value of $\delta = .1$ resulted in a "V" cycle, (1,2,3,2,1), which gave slightly inferior results.

Test 2 Equation (4) is solved on Grid 2 with Dirichlet boundary conditions at all boundaries. Grid 2 is meant to represent an airfoil with a flap. Alternating Zebra smoothers were used on all component grids, since they all are stretched in both the r and s directions. A value of $\eta = .36 = (.6)^2$ was chosen since the alternating smoothers are comprised of two sweeps.

Convergence results for the two tests are summarized in table A2.1. r(p) is the residual on the finest composite mesh after the pth multigrid iteration.

$$\mathrm{r}(p) = \|\mathbf{f}^1 - \mathbf{A}^1\mathbf{v}^1(p)\|$$

Define $WU(p)$ to be the number of work units used for the pth iteration. A work unit is the amount of work to perform one iteration of SOR on the composite mesh. The effective convergence rate, ECR, is defined as

$$\mathrm{ECR}(p) = \left(\frac{\mathrm{r}(p)}{\mathrm{r}(p-1)}\right)^{\rho} \quad , \quad \rho = (WU(p))^{-1}.$$

Theoretically the ECR for multigrid should be independent of the grid spacing h as $h \to 0$. In contrast the (effective) convergence rate for many other standard iterative schemes deteriorates as $h \to 0$. For optimal SOR, ECR $\approx 1 - c_1 h$, and for Gauss–Seidel, ECR $\approx 1 - c_2 h^2$. In table A2.1 the ECR increases on the fourth iteration for Grid 1, since the solution has almost converged to single precision accuracy.

In tables A2.2 and A2.3 the errors in the solution to the elliptic problem are given. The problem was solved on the fine mesh (level 1) and also on each of the coarser meshes. The error on level l, $e(l)$, is the maximum difference between the calculated and true solution. The number of points on the component grids in the r and s directions is given by n_r and n_s, respectively. Not all the points are used, since there is overlap between the grids. Level 1 of Grid 1 has 8431 computational points while there are 7431 computational points on the finest level of Grid 2. Contour plots of the calculated solutions are shown in figures A2.1 and A2.2.

l	k	n_r	n_s	$e(l)$	$e(l)/e(1)$
1	1	93	77	1.5×10^{-3}	1.
	2	81	21	1.1×10^{-3}	1.
	3	81	21	1.1×10^{-3}	1.
	4	93	17	1.7×10^{-3}	1.
2	1	47	39	7.4×10^{-3}	5.
	2	41	11	5.4×10^{-3}	4.8
	3	41	11	5.4×10^{-3}	4.8
	4	47	9	7.4×10^{-3}	4.4
3	1	24	20	3.7×10^{-2}	25.
	2	21	6	2.6×10^{-2}	23.
	3	21	6	2.6×10^{-2}	23.
	4	24	5	3.3×10^{-2}	20.

Table A2.2: Errors in Solution for Grid 1

l	k	n_r	n_s	$e(l)$	$e(l)/e(1)$
1	1	101	61	1.8×10^{-2}	1.
	2	121	13	2.0×10^{-2}	1.
	3	111	13	5.7×10^{-3}	1.
2	1	51	31	9.8×10^{-2}	5.4
	2	61	7	8.7×10^{-2}	4.4
	3	56	7	2.7×10^{-2}	4.7

Table A2.3: Errors in Solution for Grid 2

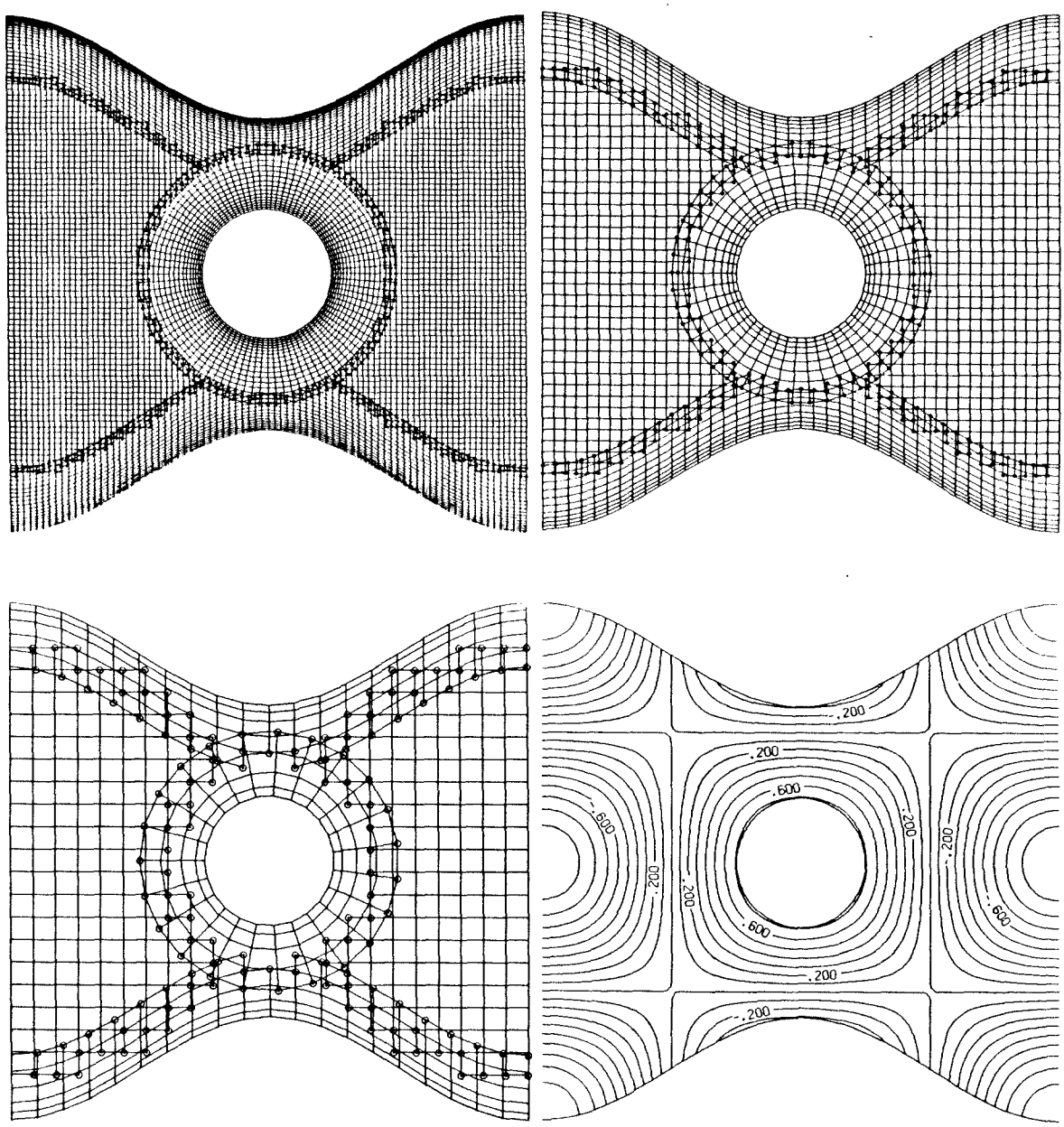


Fig. A2.1. Grid 1: Composite meshes for multigrid levels 1, 2 and 3 and a contour plot of the calculated solution.

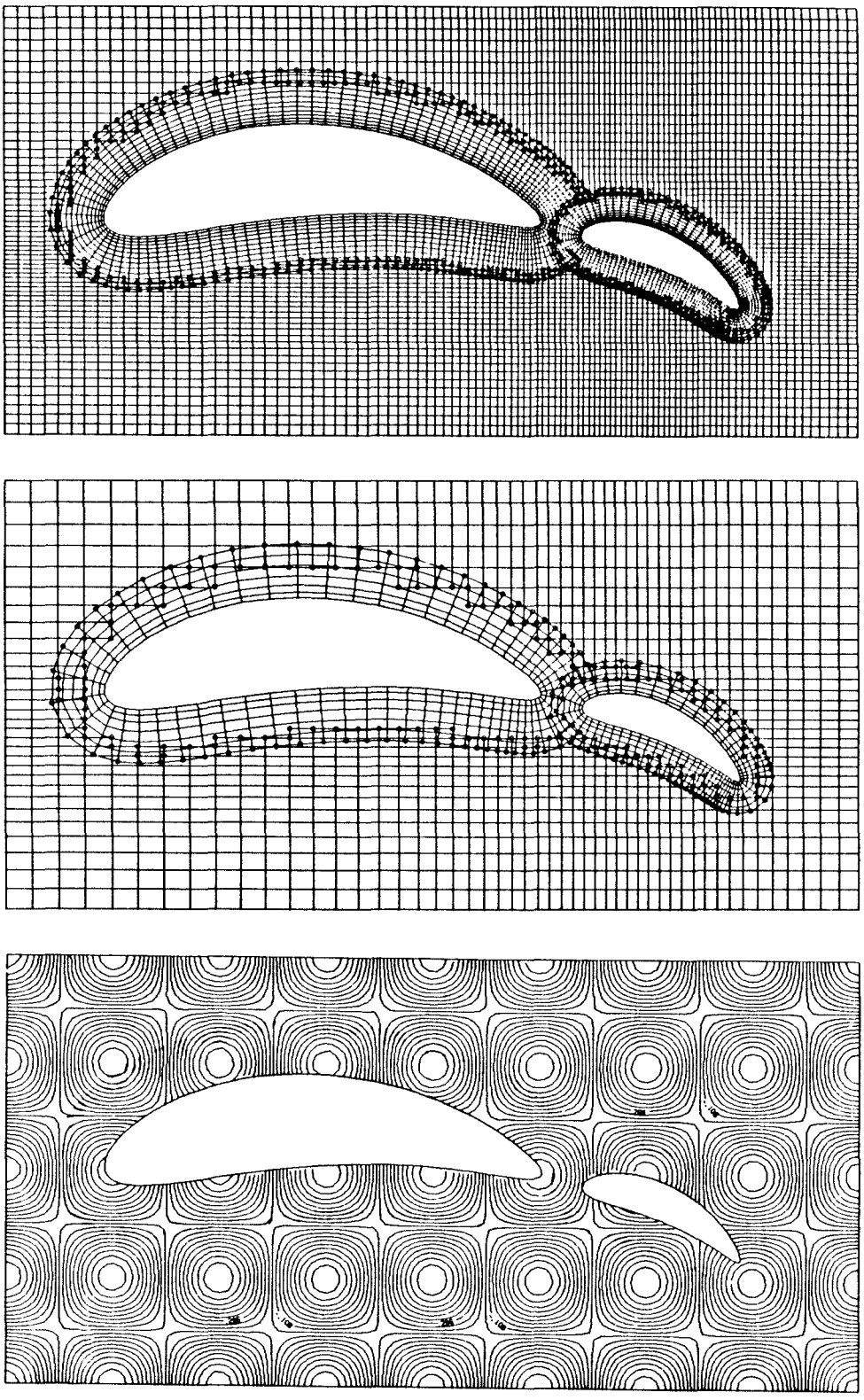


Fig. A2.2. Grid 2: Composite meshes for multigrid levels 1 and 2 and a contour plot of the calculated solution.

References

[1] E. H. Atta and J. Vadyak, *A Grid Overlapping Scheme for Flowfield Computations about Multicomponent Configurations*, AIAA J., **21** No. 9, (1983), pp. 1271–1277.

[2] J. A. Benek, J. L. Steger and F. C. Dougherty, *A Flexible Grid Embedding Technique with Application to the Euler Equations*, AIAA Paper No. 83–1944, Danvers, Mass., 1983.

[3] M. Berger, *On Conservation at Grid Interfaces*, ICASE Report No. 84–43, 1984.

[4] J. U. Brackbill and J. S. Saltzman, *Adaptive Zoning for Singular Problems in Two Dimensions*, J. Comp. Phys., **46** (1982), pp. 342–368.

[5] A. Brandt, *Guide to Multigrid Development*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin (1982), pp. 220–312.

[6] G. Chesshire, *CMPGRD: A Composite Grid Construction Program*, To appear.

[7] S. A. Coons, *Surfaces for Computer-Aided Design of Space Forms*, Dept. of Mech. Eng., Mass. Inst. of Tech., Cambridge, 1964.

[8] P. R. Eiseman, *A Multi-Surface Method of Coordinate Generation*, J. Comp. Phys., **33** (1979), pp. 119–151.

[9] P. R. Eiseman, *Grid Generation for Fluid Mechanics Computations*, Ann. Rev. Fluid Mech., **17** (1985), pp. 487–522.

[10] S. K. Godunov and G. P. Prokipov, *The Use of Moving Meshes in Gas-Dynamical Computations*, USSR. Comput. Math. and Math. Phys., **12** (1972), pp. 299–319.

[11] S. C. Eisenstat, M. C. Gursky., M. H. Schultz and A. H. Sherman, *Yale Sparse Matrix Package*, Research Reports 112 and 114, Yale University, Dept. of Computer Science, May 1977.

[12] W. D. Henshaw, *Part I: The Numerical Solution of Hyperbolic Systems of Conservation Laws; Part II: Composite Overlapping Grid Techniques*, Ph.D. thesis, Department of Applied Mathematics, California Institute of Technology, 1985.

[13] W. D. Henshaw and G. Chesshire, *Multigrid on Composite Meshes*, submitted to SIAM J. Sci. Stat. Comput.

[14] B. Kreiss, *Construction of a Curvilinear Grid*, SIAM J. Sci. Stat. Comput., **4** No. 2 (1983), pp. 270–279.

[15] C. W. Mastin and J. F. Thompson, *Elliptic Systems and Numerical Transformations*, J. Math. Anal. App., **62** (1978), pp. 52–62.

[16] L. G. M. Reyna, Ph.D. thesis, Department of Applied Mathematics, California Institute of Technology, 1982.

[17] G. Starius, *Construction of Orthogonal Curvilinear Meshes by Solving Initial Value Problems*, Numer. Math., **28** (1977), pp. 25–48.

[18] G. Starius, *Composite Mesh Difference Methods for Elliptic Boundary Value Problems*, Numer. Math., **28** (1977), pp. 243–258.

[19] G. Starius, *On Composite Mesh Difference Methods for Hyperbolic Difference Equations*, Numer. Math., **35** (1980), pp. 241–255.

[20] K. Stüben and U. Trottenberg, *Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin (1982), pp. 1–176.

[21] J. F. Thompson, F. C.Thames and C. W. Mastin, *Automatic Numerical Generation of Body-Fitted Curvilinear Coordinate System for Field Containing Any Number of Arbitrary Two-Dimensional Bodies*, J. Comp. Phys., **15** (1974), pp. 299–319.

[22] J. F. Thompson, ed., *Numerical Grid Generation*, Elsevier, New York, 1982.

[23] A. M. Winslow, *Numerical Solution of the Quasilinear Poisson Equation in a Nonuniform Triangle Mesh*, J. Comp. Phys., **2** (1967), pp. 149–172.

[24] P. Woodward and P. Colella, *The Numerical Simulation of Two-Dimensional Fluid Flow with Srong Shocks*, J. Comp. Phys., **54** (1984), pp. 115–173.