# Automatic contouring by piecewise quadratic approximation.

Thomson, G. D.

*Award date:*
1984

*Awarding institution:*
University of Bath

Link to publication

# AUTOMATIC CONTOURING BY PIECEWISE QUADRATIC APPROXIMATION

submitted by G.D. Thomson

for the degree of Ph.D.

of the University of Bath

1984

[illegible]

G.D. Thomson

CONTENTS Page

## SUMMARY

In this thesis we introduce and develop a new method for the automatic contouring of smooth surfaces, which produces high quality results at relatively low cost.

We begin (Chapter 1) by reviewing the contouring methods in the literature; serious limitations are revealed which appear to justify a search for a new and better method. In Chapter 2 a seamed quadratic finite element is introduced which is suitable for approximating smooth functions whose values and gradients may be evaluated at the nodes of a rectangular grid. We suggest using an efficient published subroutine due to Marlow and Powell (1976) or some suitable alternative to plot the contours of the approximant surface; the resulting method produces accurate and visually smooth contours at relatively little expense.

Chapter 3 is an explanation of CONICON, the Fortran subroutine package which implements this contouring method. In Chapter 4 CONICON is used to contour surfaces arising from a variety of applications.

Chapter 5 describes an error analysis of the seamed quadratic element which enables us to obtain bounds for the error involved in using the element to approximate a function.

In Chapter 6 we implement an extension of our contouring method which uses local subdivision of elements in order to reduce local variations in the error involved in contouring a function. Various possible criteria for splitting are suggested, some of which are tested on known functions.

Finally (Chapter 7) we conduct a (fairly superficial) comparison of a number of contouring packages which are currently available. We

discuss the features which each package offers and attempt to assess their quality, simply on the evidence available from user documentation and advertising literature.

Reference

MARLOW, S. and POWELL, M.J.D. (1976). A Fortran subroutine for plotting the part of a conic that is inside a given triangle. *UKAEA Harwell Paper AERE-R 8336,* HMSO London.

## ACKNOWLEDGEMENTS

I should like to record here my deep gratitude to my supervisors, Robin Sibson and Bernard Silverman, for all the friendly assistance which they have given in guiding me through the course of my research, and in particular for a large number of stimulating and invaluable discussions. The project's successful completion is due in no small part to their support and enthusiasm.

I also wish to thank John Fitch, Julian Padget, John Slape and especially Adrian Bowyer for their help with computational aspects of the project; and Chris Dorling, David Hocken, M.C. Jones, Wyn Lotwick, Clive Osmond, Brian Woodward and numerous other colleagues and friends for helping to make my time at Bath an enjoyable one, and for encouragement during the seemingly endless 'writing up' stage.

Persons from organisations outside the University of Bath to whom I am particularly indebted are Ann Francescon, Jens Daabeck and Dr. M.A. Sabin.

The Social Science Research Council deserves my thanks for its funding of the whole enterprise; and the European Centre for Medium-range Weather Forecasts for generously allowing me to report work carried out while under its employment. I am also grateful to Kansas Geological Survey, Numerical Algorithms Group and John Wiley and Sons for granting me permission to reproduce published illustrations whose respective copyrights they hold; and to Applied Research of Cambridge Ltd for the use of its computer drafting system GDS in the preparation of diagrams.

Finally I wish to thank my typist Jacqui Walker for her accurate and obliging preparation of the script.

# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction to contouring; definitions and assumptions

In this thesis we shall consider the problem of constructing contour maps of smooth surfaces automatically, with the aid of digital computers and associated graphics devices. Particular attention will be paid to the aim of achieving a high quality of output without incurring inefficiency in the use of computer resources.

The history of automatic contouring is a short one:- before the beginning of the 1960s, contouring was necessarily a long laborious process carried out by hand by skilled draftsmen; but the advent of computer graphics provided the opportunity to reduce the typical construction time for a contour plot from several hours to a few minutes or less, and at the same time to introduce a degree of scientific objectivity into a process which had previously been subject to the whims and prejudices of the individual draftsman.

At the outset of this project nearly twenty years had passed since the publication of the first automatic contouring algorithms, but the state of the art was still not satisfactory: though several contouring methods had been suggested and implemented, none was capable of combining high quality results with efficient use of resources. Consequently it was felt that further attempts to discover improved methods would be fully justified, particularly in view of the considerable importance of the contouring problem.

This importance is a consequence of the generality and widespread applicability of the problem; a contour map is an attempt to represent a three dimensional surface in two dimensions, and is simply a higher dimensional analogy of a curve or graph. Though it is not the only such means of displaying a surface (probably the most familiar alternative method is the perspective block diagram), it has the distinction that it may readily be combined with maps of related data (for example geographical features such as coastlines and political boundaries), and for this reason it has long been the most frequently used and best understood means of surface display. This is not to berate some of the alternative methods, which are sometimes to be preferred as a means of providing an instantly cognizable qualitative impression of the nature of a surface; however the contour map has undoubtedly superior qualities from a quantitative point of view.

Applications of contouring occur in virtually all scientific disciplines, and it would be pointless to attempt to list them comprehensively:- probably the most familiar application areas are relief mapping in cartography (though surfaces encountered in this area are not always smooth and are therefore not wholly amenable to contouring by the methods discussed in this thesis) and plotting of isobars, isotherms, etc in meteorology. The oil exploration industry is another very important application area, several contouring packages having been designed specifically for use in such applications. In Chapter 4 we shall consider examples from a few of the areas in which contouring can be beneficial to the scientist.

We begin though by defining precisely what is meant by contouring and by making some assumptions.

Our first assumption is that the function $f(x, y)$ which represents the surface which we wish to display is $C^1$ i.e. it has continuous first derivatives - and this is what will be implied when a surface is referred to as smooth. Our function is therefore prohibited from exhibiting either 'cliff edges' or 'ridges' (discontinuities in the function itself and its first derivatives respectively). However, higher order discontinuities, which are not normally detectable by eye, will be permitted.

Secondly the surface is assumed to be a known function f of the dependent variables x and y throughout the region of interest. This is not such a restrictive assumption as it might at first appear: it is true that in many applications the form of function being contoured is unknown and that typically we are faced with a finite number of surface measurements taken at irregularly scattered data sites (for example these might correspond to the sitings of weather stations or balloons); however in such circumstances several methods of interpolation are available to the scientist which will enable him to construct a smooth surface passing exactly through all known points, which can be evaluated anywhere within the 'window' or region of interest. This interpolant then becomes the known function which may be displayed by contouring. As an alternative to interpolation, on those occasions when measurements are subject to error, the investigator may opt to carry out a process of smoothing, which does not fit the surface to the original data values exactly, but trades off goodness of fit of the function against some appropriate measurement of its smoothness.

In either case it is important to emphasize that we regard the surface-fitting process as being entirely separate from the contouring process. Many authors in the literature have attempted to merge the two into a single process, but such efforts invariably result in restricting the quality of performance of both.

Our final assumption is that the surface is never constant (that is, its first derivative is never zero) over a region of finite area. Stationary points are of course not prohibited by this assumption.

The contour of the function f at level h is then defined simply as

$$\{(x, y);\ f(x, y) = h\} \tag{1.1}$$

Properties of contours follow immediately from the following well-known theorem in analysis:-

IMPLICIT FUNCTION THEOREM (stated without proof)

Let $f(x, y)$ have continuous derivatives $f_x$ and $f_y$ in a neighbourhood of a point $(x_o, y_o)$, where

$$f(x_o, y_o) = 0, \quad f_y(x_o, y_o) \neq 0 \tag{1.2}$$

Then, centred at the point $(x_o, y_o)$, there is some rectangle

$$x_o - \alpha \leq x \leq x_o + \alpha, \; y_o - \beta \leq y \leq y_o + \beta \tag{1.3}$$

such that for every $x$ in the interval I given by $x_o - \alpha \leq x \leq x_o + \alpha$, the equation $f(x, y) = 0$ has exactly one solution $y = g(x)$ lying in the interval $y_o - \beta \leq y \leq y_o + \beta$. This function g satisfies the initial condition $y_o = g(x_o)$ and, for every x in I,

$$f(x, g(x)) = 0 \tag{1.4}$$

$$y_o - \beta \leq g(x) \leq y_o + \beta \tag{1.4a}$$

$$f_y(x, g(x)) \neq 0 \tag{1.4b}$$

Furthermore, g is continuous and has a continuous derivative in I, given by the equation

$$y' = g'(x) = -\frac{f_x}{f_y} \tag{1.5}$$

As a consequence of this theorem, contours of any surface satisfying our assumptions will be smooth lines with no visible 'corners'. Contours (at the same level) cannot cross each other except at stationary points where the Theorem breaks down (such behaviour is investigated in Chapter 2); contours at different levels can of course never touch or cross each other.

## 1.2 A word on interpolation methods

The field of interpolation and smoothing is a wide one, which it is possible only to touch upon within the context of this thesis. Smoothing is a particularly difficult problem which is still not very satisfactorily developed computationally, and we shall choose to ignore it for the remainder of this thesis. However, as the raw data which we use for contouring often arise in the form of measurements at irregularly distributed locations in space, we shall devote the current section to a very brief discussion of some of the interpolation methods available.

Sibson (1982) provides a highly readable introduction to the subject, and lists those properties which he considers desirable in any interpolation method. These may be summarised as follows:-

* The function must be at least continuously differentiable; higher order continuity is less important as this does not appear to be detectable by eye, except in special cases.
* The function should run as little risk as possible of provoking misinterpretation of the data.
* The data sites should occupy most of the window; their precise setting should not matter, and the interpolant should have at least a continuous response to data site position i.e. the interpolant should not jump from one state to another in response to a small change in data site position.
* The method should not depend on arbitrary choices unrelated to the data e.g. choice of coordinate system, number of points influencing the interpolated value at any location, etc.
* The dependence of the interpolant function on the data values should be reasonably well behaved and simple: if possible it should be linear, so that if the system of data values is multiplied throughout by a scalar, the interpolant is also multiplied by that scalar, and if two systems of data values at the same data sites are added, the interpolant for the sum should be the sum of the interpolants.
* The interpolant should be localised, in that in some suitable sense only data sites which are reasonably near neighbours should influence the interpolated value at a point.

* The method should be directly computationally feasible on a reasonably large scale - Sibson suggests 10,000 data sites and interpolation to as many points without undue difficulty.

* It is attractive if the theory generalises to n dimensions.

* It is helpful in applications if the gradient, as well as value, of the interpolant is calculable.

* We would expect the interpolation method to recover exactly functions from some simple class - a method which could not recover linear functions at least would be unacceptable. However in general a tradeoff will exist between the degree of localisation and the complexity of the class of recoverable functions.

A host of alternative methods for interpolation exist in the literature, several of which have been implemented in practice; however few, if any, satisfy all the criteria outlined above, and many fail to conform to a number of them. It is very common to find, for example, that a truly local method has been rendered so only by forcing the user to make one or more arbitrary decisions regarding the number and means of selection of those neighbouring surface measurements which influence the value of the interpolant at a point.

Sibson regards only three standard classes of interpolation method as worthy of serious discussion; finite element methods, kriging and stiff lamina methods.

Finite element methods (see for example Lawson (1977)) split up the window into polyhedral panels in a data-determined manner, and fit together smooth functions (finite elements) across the panels. Such methods can be very efficient, but the major disadvantage of all such methods is their discontinuous response to data site position; and it is particularly awkward that discontinuities occur, which must be resolved artificially, when the data sites are partially or wholly on a regular rectangular grid.

Kriging methods (see for example Delfiner and Delhomme (1975)) present interpolation as a process of statistical estimation within the context of an elaborate stochastic model based on spatial moving averages. Sibson contends that such a model is usually not well-related to the phenomena it purports to describe. He also notes that many unresolved questions remain regarding the degree of smoothness of the interpolant functions and the behaviour of their gradients, and 'the computational position appears to be highly unsatisfactory'.

Finally Sibson considers stiff lamina methods (see Wahba and Wold (1975), Wahba (1979)). In two dimensions, at the lowest degree of smoothness the physical model is a uniform stiff lamina constrained to take the data values as (infinitescimal) displacements at the data sites. Like kriging, these methods are not localised, and this results in major computational difficulties. Wahba reports successful results working with 120 data sites, but warns of the difficulty of attacking substantially larger problems.

Sibson presents his own method, Natural Neighbour Interpolation, as an alternative to the existing methods. The method is based on the Dirichlet Tessellation (see for example Green and Sibson (1979)), a geometrical construction which divides the window into 'tiles', regions within which all points share a common nearest data site. The tessellation is used by Natural Neighbour Interpolation as a means of defining the 'neighbours' of any point in a natural and completely data-determined manner. The method also succeeds in satisfying all the desiderata for a good interpolation method which were set out above. For these reasons, and because the author had the good fortune to have access to the TILE software package in which the method is implemented, the $C^1$ Natural Neighbour method has been used in all examples in this thesis which required interpolation to be carried out prior to contouring.

## 1.3 Contouring methods in the literature; 'contour following' methods

In general contouring methods in the literature may be divided into two categories:- 'contour following' methods, which attempt to contour directly, following the true contours of the known function; and approximation methods, which replace the function f by a function $\hat{f}$ of a simpler form, whose contours are relatively easy to plot directly. In this section and the next we shall examine these two categories of method in some detail.

In the course of these sections, and indeed throughout the thesis, it should be borne in mind by the reader that currently most graphics devices are capable of plotting straight lines only, and it is therefore necessary to approximate a curved line by a sequence of straight lines using software. As long as the lines forming this approximation are sufficiently short the approximation will be undetectable to the eye. However the use of excessively short straight line segments is inefficient. The goal which we are therefore aiming for is to achieve an approximation which is good enough to deceive the eye but uses as small a number of straight line segments as possible. This implies that the segments should be of variable length, and that length should be inversely related to contour curvature.

### 'CONTOUR FOLLOWING' METHODS

These methods take the most direct approach to contouring a function, following each contour along, point by point, interpolating from values close to the track of the contour to determine its position.

Variations are described by Batcha and Reese (1964), Lodwick and Whittle (1970), Falconer (1971) and Schagen (1982). Leaving aside a number of details, the principle behind these methods may be summarized by Figure 1.1. Suppose plotting of the contour has begun, and the last straight line

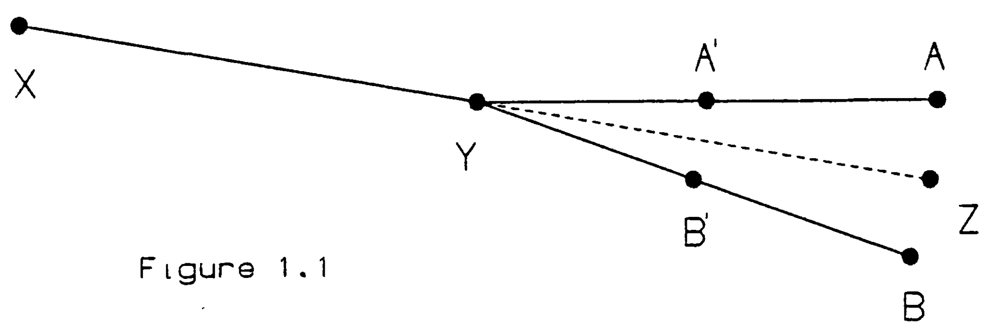
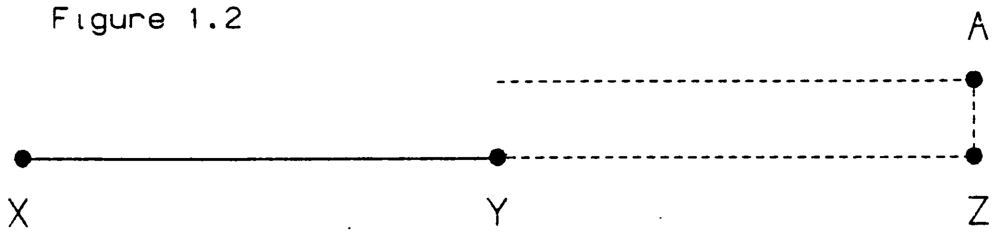


Figure 1.1

segment on the contour is the line XY. The next point is determined by making a pair of probes, at A and B, where A and B are say 10% to the left and right of the point Z (2Y-X in vector notation). If the surface heights at A and B are on opposite sides of the contour level, inverse linear interpolation is used to derive the next point on the straight line between A and B. If however they are on the same side, then A is replaced by A' = (A + Y)/2 and B by B' = (B + Y)/2 and the test repeated. Step length is therefore to some extent adaptive. However this basic method is clearly not robust: the point Y does not in general lie precisely on the contour and consequently there is a danger that the true contour may not enter the sector bordered by the lines AY and YB; the method must therefore be complicated by the introduction of a safety modification of some kind.

An alternative strategy, suggested by Sabin (1980) (see Figure 1.2) evaluates the surface height at the point Z and then, depending on whether

Figure 1.2



this is above or below the contour, at a fixed distance to the left or right. The smaller this fixed distance is, the greater will be the visual smoothness of the contours.

If the second probe point is on the same side of the contour as the first, then instead of halving the scale of the probe, iterated inverse interpolation is used along a line parallel to the previous step to locate the next point. The next step will then automatically be shorter, so that as curvature increases the step length shortens appropriately.

If the second probe point is on the opposite side of the contour, inverse linear interpolation gives the next point as in the previous example, and the fact is used to indicate that the step size for the next probe can be increased.

However some serious problems associated with this class of method remain which have never been satisfactorily solved:- firstly it is by no means obvious how to locate a starting point on every separate section of contour, and to avoid finding more than one starting point on thc same section. Lodwick and Whittle (1970) and Falconer (1971) both advocate the use of a regular grid to locate starting points, but the effectiveness of such a method is obviously restricted by the coarseness of the grid. Sabin suggests computing all stationary points and then constructing a spanning tree joining all of these by straight lines, cutting all contours at least once. Iterated linear interpolation could then be used to give the starting points. Though theoretically attractive, such an approach would be difficult to implement in practice and would impose severe limitations on the degree of generality of the method, and the ease of its use by non-specialists.

Another major problem associated with these methods is that of identifying when the end of a closed loop has been reached, since return to the exact starting point of the contour is extremely unlikely (and in an area of relatively low contour curvature it may be possible to overshoot by a considerable distance). The danger of tracing a contour section more than once from a single starting point therefore arises, and this problem has not been dealt with satisfactorily in the literature.

Finally, if evaluation of the function f at an arbitrary point is a relatively expensive operation then contour-following methods can turn out to be very costly, as they require a relatively large number of surface evaluations to be made in the course of contour construction. This number will increase in proportion with the number of contour levels plotted.

For these reasons, contour-following methods have met with very little practical success; nearly all widely used contouring packages employ methods of the type discussed in the following section.

## 1.4 Contouring by surface approximation

The other important category of contouring methods is based around the idea of approximating the (relatively complicated) known function f by a simpler function $\hat{f}$ whose contours may be followed directly with little difficulty. In practice this has meant that f has been approximated by piecing together linear or quadratic functions only, since the difficulties of contouring polynomials increase very considerably with their complexity. We shall begin by examining the simpler of the two, piecewise linear methods, which are still probably the most frequently used class of automatic contouring methods.

### 1.4.1 Piecewise linear contouring

The simplest imaginable function of two variables (constants excluded) is a linear function or plane, and the contours of such a function, being straight lines, may be plotted trivially. For this reason the most popular means historically of contouring a function has been to approximate that function by piecing together planar sections and to plot the contours of the resulting piecewise linear surface.

Several variations of such an approach have been taken both in the literature and in practice. In cases where the data sites are scattered irregularly in space a common strategy is to construct a triangulation of the area of interest (see for example the Delaunay triangulation (Green and Sibson, 1979)) with the data sites lying at the vertices of that triangulation. A unique piecewise linear surface of triangular 'patches' is then defined. However this is one of the most primitive contouring methods imaginable, and like other methods which attempt to merge the processes of interpolation and contouring has a number of inherent defects. To begin with, it only permits contouring within the convex hull of the data sites while in practice it is usually desirable to contour throughout a rectangular window containing all the data sites.

The appearance of the contours themselves is a matter of still greater concern: in areas where data are sparse the triangular patches are necessarily large and consequently contours often exhibit pronounced 'corners' along the boundary lines between triangles. Such behaviour will indeed occur, though usually less markedly, wherever a contour crosses the boundary line between a pair of triangles and is a property of all piecewise linear methods - it follows from the discontinuous nature of the first derivatives of the piecewise linear approximant along triangle edges. If is also worth noting that one can often identify quite clearly the locations of the data sites simply from viewing the pattern of contours - obviously this can never be considered an accurate likeness of the true underlying surface.

Further problems arise in the implementation of methods of this sort: in practice we usually wish not to plot the contours of each triangular patch individually, but to trace each contour without break (except possibly for labels) from start to end, in order to minimise pen movement (an important consideration on a pen plotter) and to facilitate contour annotation (a necessary addition in most applications). This can be done in either of two ways:- (a) by contouring each triangle individually and carrying out an internal matching and linking process, or, preferably (b) by following each contour along from start to end as it is constructed. The use of the method described above makes (b) a relatively difficult operation, requiring a fairly complex data structure, and it may therefore be necessary to resort to (a), which is generally less efficient.

Those piecewise linear methods which do not attempt to combine the processes of interpolation and contouring, normally evaluate the known function (or interpolant) at the nodes of a regular (almost always rectangular and usually square) grid, which becomes the basis for contouring. A review and discussion of these methods and the relatively trivial differences between them is given by Sutcliffe (1980). A considerable number of authors (for example Cottafava & Le Moli (1969), Heap & Pink (1969), Rothwell (1971) and Crane (1972)) advocate an approach which is not, strictly speaking, piecewise linear in our terminology, but is very closely related.

Figure 1.3

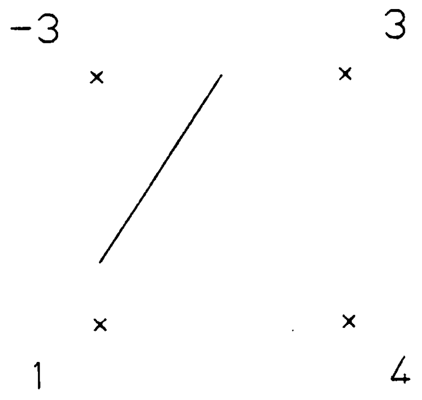
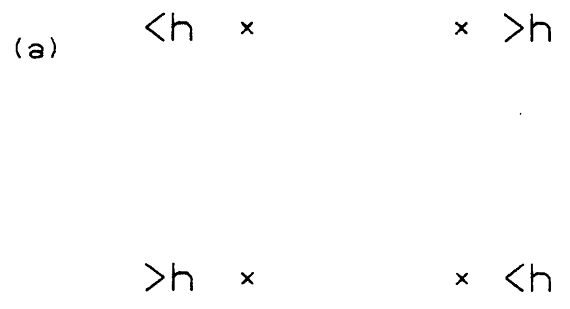
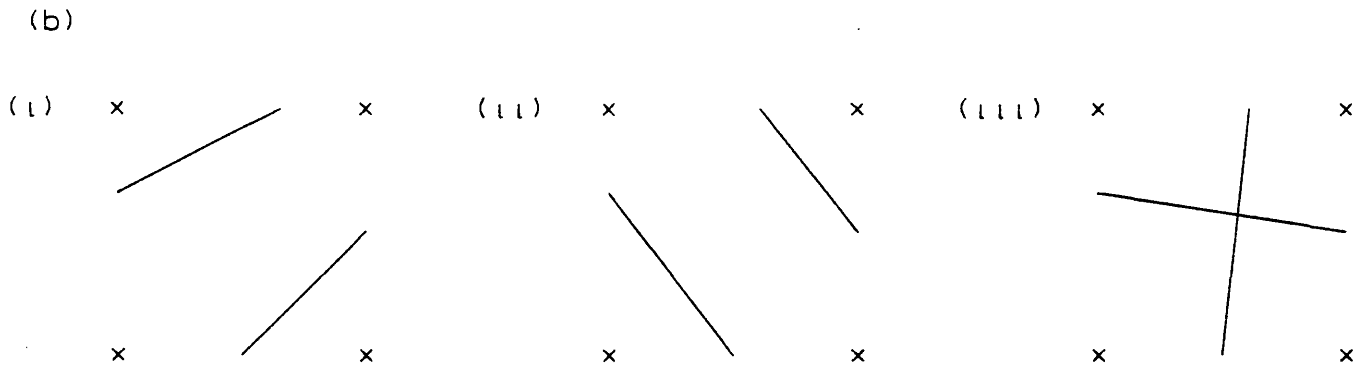


Figure 1.3 provides a simple illustration of the behaviour of such methods within a single rectangular grid cell. The points where the contour (in this case at level 0) intersects the edges of the cell are calculated by

inverse linear interpolation and then linked by straight lines. However this is not a true linear fitting of the function values at the vertices, since linear functions having three free parameters cannot in general be fitted to four data values. Consequently degeneracies must occur, and this happens whenever data of the type indicated by Figure 1.4(a) are encountered.

Figure 1.4
(contour level = h)





In cases of this type inverse linear interpolation yields four intersections between the contours and the cell's edges. We are therefore faced with three possible outcomes, all of which fit correctly, and it is impossible to determine on the basis of the available data which is the true solution (though case (iii) occurs with probability zero). In the literature a number of alternative proposals for the systematic resolution of degeneracies of this type have been offered:- examples are a 'high ground on the right' rule (Heap & Pink (1969)) and the slightly less arbitrary idea of choosing the possibility which in some sense minimises the change in contour direction (Robinson & Scarton (1972)). However the only solution considered by the present author to be of any real worth is one advocated by Dayhoff (1963) and others; this is to make a further evaluation (or more normally an estimate) of the function value at the centre of the rectangle and to divide the rectangle into four triangles (Figure 1.5) which may be contoured without ambiguity. Indeed this author's personal

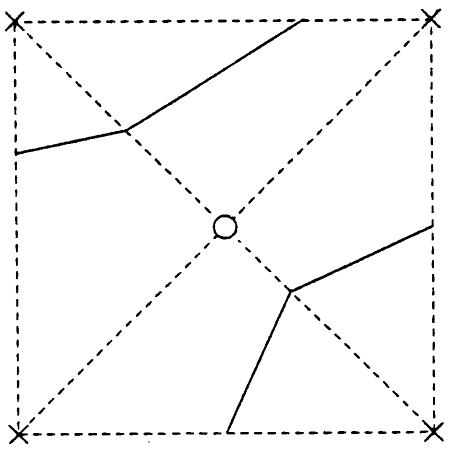


Figure 1.5

preference is to carry out such a process in all grid cells regardless of whether a degeneracy has occurred; if the central value is estimated by averaging the values at the vertices then this is an inexpensive operation, and besides increasing contour smoothness by reducing the length of straight line segments it has the highly beneficial effect of producing a true piecewise linear approximation of the underlying surface. An important consequence of this, though one which is often overlooked in the literature, is that the contouring method may then be subjected to a proper error analysis which will enable the user to obtain bounds for the error involved in using the piecewise linear approximation. Though an analogous but considerably more difficult task has been carried out successfully (see Chapter 5) in the case of the piecewise quadratic element which will be introduced in Chapter 2, this does not appear to have been attempted in the literature in the context of piecewise linear methods.

Given this simple approach to constructing contours across individual grid cells, the following of contours from cell to cell and the avoidance of repetition are relatively straightforward tasks; we shall not go into the details, which are discussed adequately by Sutcliffe (1980).

We now present an example (Figure 1.6) of a surface which has been contoured by the method suggested above; the function is

$$f(x, y) = \exp(-\tfrac{1}{2}(4(x - 1)^2 + 6(y - 1)^2)) + \exp(-\tfrac{1}{2}(10(x - 2)^2 + 6(y - 1.3)^2 + 14(x - 2)(y - 1.3))) \tag{1.6}$$

and has been evaluated at the nodes of a 31 x 21 square grid of points (or 30 x 20 grid of cells) covering the area $\{(x, y);\ x \in [0, 3],\ y \in [0, 2]\}$. Values at the centres of grid cells were estimated by the mean of the four values at the vertices.

The major problem associated with all piecewise linear contouring is immediately apparent from this illustration: although the function is a smooth one and the grid reasonably fine, the contours of the approximation display sharp angularities in areas where the curvature of the true contours is highest, and in very few areas of the plot can they truly be described as 'visually smooth'. The smoothness of the map can of course be improved by choosing a finer grid; however this results in increased use of both CPU and memory and in this example it was found that contouring over a 120 x 80 grid of cells, a very costly operation, was still not sufficient to create an impression of smoothness at the highest contour on each peak.

The function which we have considered in this example is considerably less complex than many of those functions which are encountered in practical applications. If a satisfactory degree of smoothness is to be achieved in practical examples the use of piecewise linear contouring methods must therefore incur considerable expenses in computer resources, and these expenses are unacceptably high in many applications.

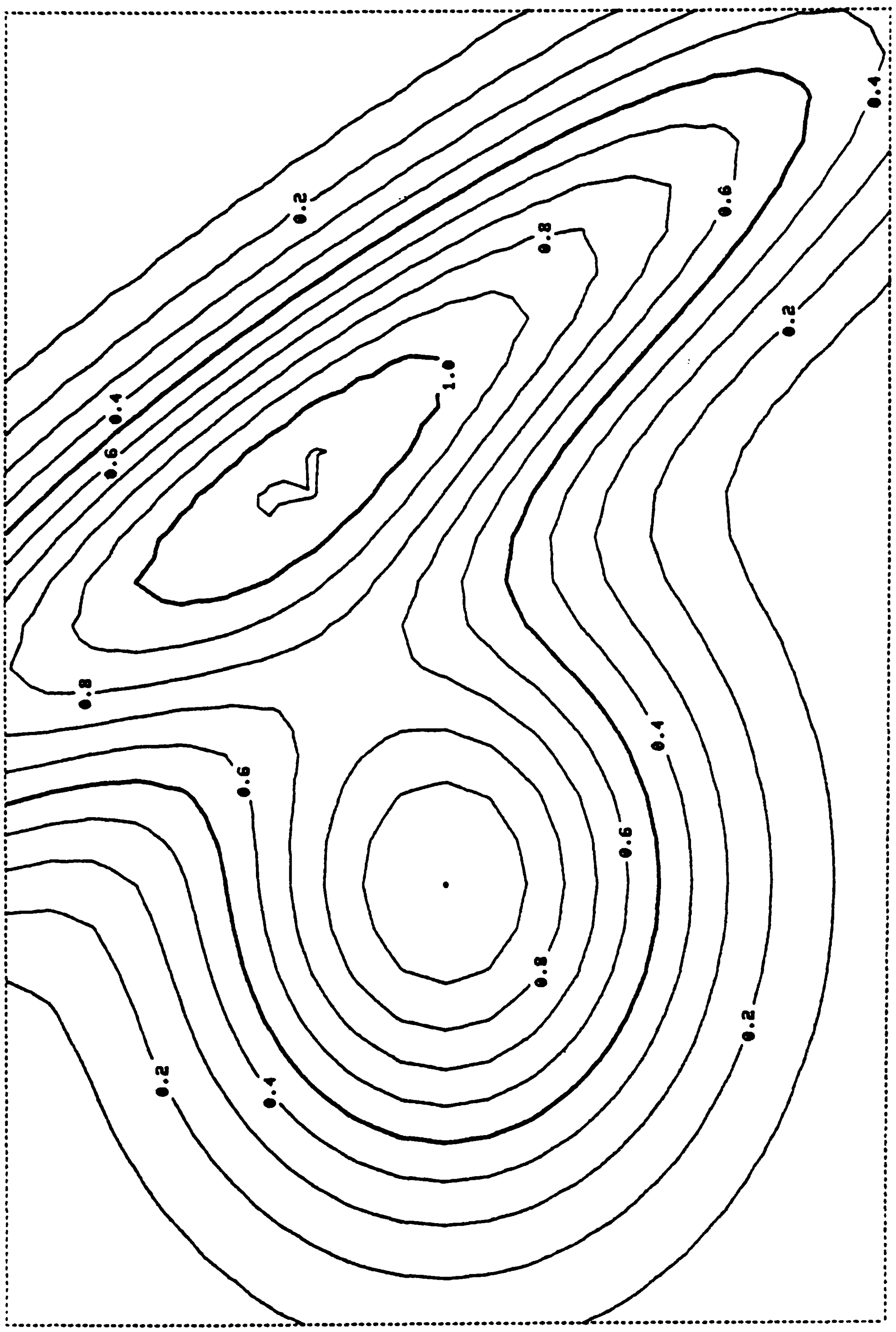


Figure 1.6 An example of contouring by piecewise linear approximation (30 x 20 grid of cells).

In an attempt to solve this problem many authors have suggested the use of a relatively coarse grid to define contour 'vertices', followed by the application of a curve-fitting algorithm to smooth the contours. A multitude of curve-fitting algorithms appear in the literature, many of which have been suggested as suitable for contour smoothing (e.g. McConalogue (1970, 71), Butland(1980)). However, all suffer from the same crucial flaw:- because they take no account of the 3-dimensional nature of the problem, none can guarantee that contours at different levels will not cross each other. Such behaviour is particularly prone to occur near saddle points, and is of course totally unacceptable. Instead it makes much more sense to look for an approximation whose contours are themselves smooth and are relatively easy to plot.

## 1.4.2 Piecewise quadratic contouring

The function which most readily lends itself to contouring in this way is the quadratic function: though finding ways of piecing together quadratic functions in a suitable manner is a non-trivial affair, once this has been accomplished it is then possible to follow the contours of each quadratic relatively easily by expressing them in parametric form. This does not apply to many other types of function: as the complexity of a function increases the difficulty of following its contours grows very considerably:- suppose for example we take a biquadratic function of the form,

$$f(x, y) = \sum_{i=0}^{2} \sum_{j=0}^{2} a_{ij} x^i y^j \qquad (1.7)$$

a function only slightly more complicated than a quadratic. Consider the special case $f(x, y) = x(1 - x) y(1 - y)$. This clearly has a 'noughts and crosses' grid as the zero contour, and contours close to level zero will approach this pattern, comprising four or five separate pieces. The author knows of no existing method which is capable of contouring directly a function of this or greater complexity; consequently as we now consider $C^1$ surface approximations for contouring we shall concentrate exclusively on piecewise quadratic approximations.

It can be shown that a regular rectangular mesh is unsuitable for piecing together quadratic functions, for once the function is known in all rectangles along the left hand and bottom edges of the approximation, the rest is then uniquely determined. Thus the approximations which we shall consider tend to be constructed from piecing together triangular panels, though we shall still prefer to contour from data lying on a rectangular grid.

A published Fortran subroutine due to Marlow and Powell (1976) conveniently contours a quadratic function across a triangle given six parametrising values at the vertices and at the midpoints of its sides. This efficient routine has the additional advantage of an adaptive step-length rule: by keeping constant the product of straight line segment length and maximum distance from the true contour, it ensures that segments are relatively short in areas of high contour curvature, and longer where curvature is low. The user controls the magnitude of this step length parameter and can therefore tune the contours produced by the routine to any desired degree of smoothness.

Taking the Marlow-Powell routine as a 'black box' for the present, we are now free to concentrate our efforts on finding the most convenient method of piecing together quadratic functions without losing continuous differentiability across panel boundaries.

Powell (1974) considers piecewise quadratic approximations of functions from height data at the nodes of a rectangular grid. He begins by considering the possibility of approximating the function f on the basis of the four-triangle-per-grid-point construction illustrated by Figure 1.7, and suggests two possible schemes of approximation based on this construction;

however the first does not in general fit the data values exactly, while

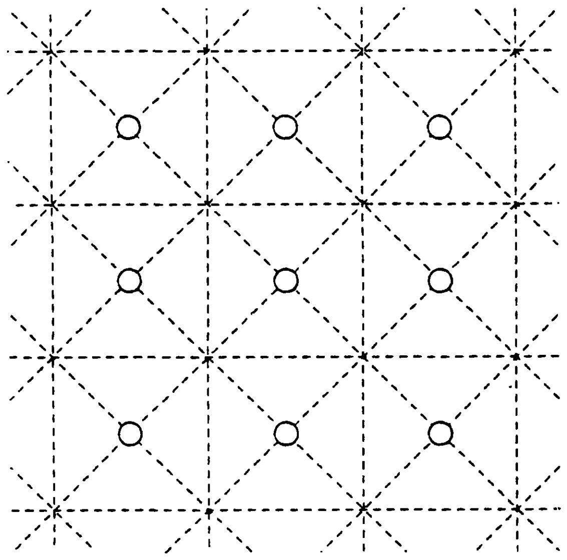


Figure 1.7

the second is not localised to the desired degree. Powell therefore goes on to present a second, eight-triangle-per-grid-point construction, which is illustrated by Figure 1.8.

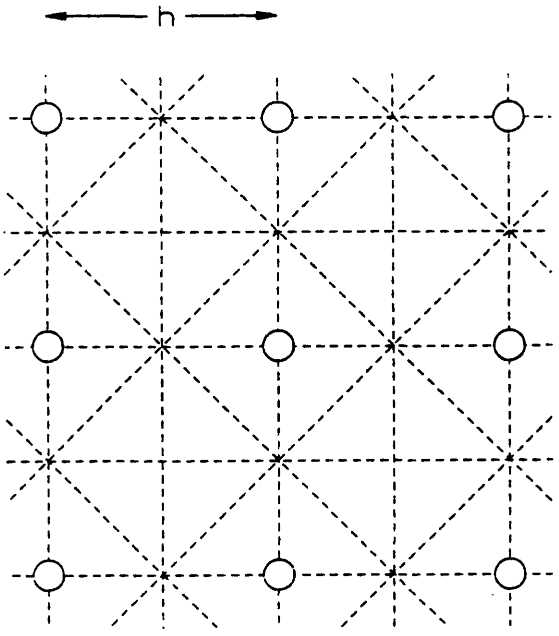


Figure 1.8

In this case he succeeds in finding a $C^1$ approximation which fits the data exactly, is truly local, (changes in the value at a grid point have no effect outside a square of side 4h centred at that grid point) and which reproduces exactly an arbitrary quadratic function - though it does suffer from mild edge effects. Powell's method has apparently met with considerable practical success (though unfortunately no examples of its use appear in his paper); however it uses value data alone and, as in multivariate optimisation, there are many attractions (particularly from the viewpoints of accuracy and localisation) in using gradient as well as value data to fit the approximant function. Gradients are available in a surprisingly high proportion of applications, and when they are not reliable methods exist for their estimation, which detract little from the advantages of using gradient as well as value data.

Attempts have therefore been made recently in the literature to discover conforming seamed quadratic finite elements:- that is, constructions normally of a triangular or rectangular shape, subdivided internally into a number of quadratic panels, which may be fitted to value and gradient data on a triangular or rectangular grid, one per cell, in such a way that continuous differentiability is preserved both within each element and across element boundaries. Finite elements are used extensively in numerical analysis and have generated a considerable volume of literature; however in such applications higher order functions possess strong advantages over quadratics from the point of view of accuracy of approximation, and consequently piecewise quadratic finite elements appear to have been neglected.

Powell and Sabin (1977) are therefore possibly the first authors to have considered the problem of constructing triangular seamed quadratic finite elements, whose quadratic panels are also triangular. Such constructions might be used to contour across a regular triangular grid, but Powell and Sabin were interested in the general case of contouring across a triangulation of the original data sites, and were thus attempting to merge the processes of interpolation and contouring.

Powell and Sabin demonstrated that the six-triangle internal sub-division illustrated by Figure 1.9 (where 0 is any point inside the

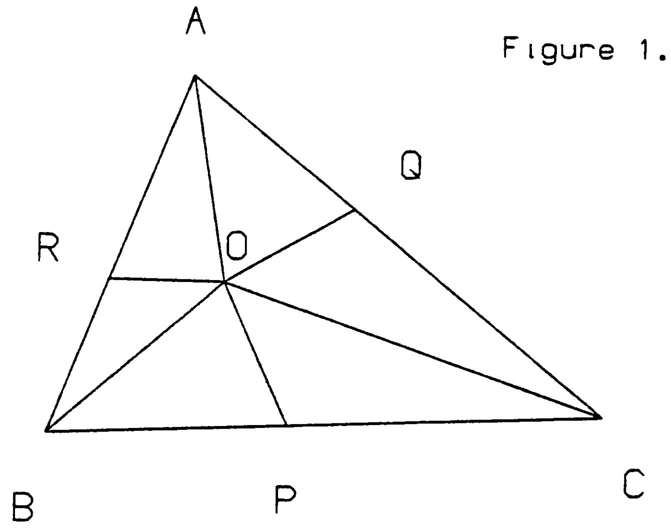


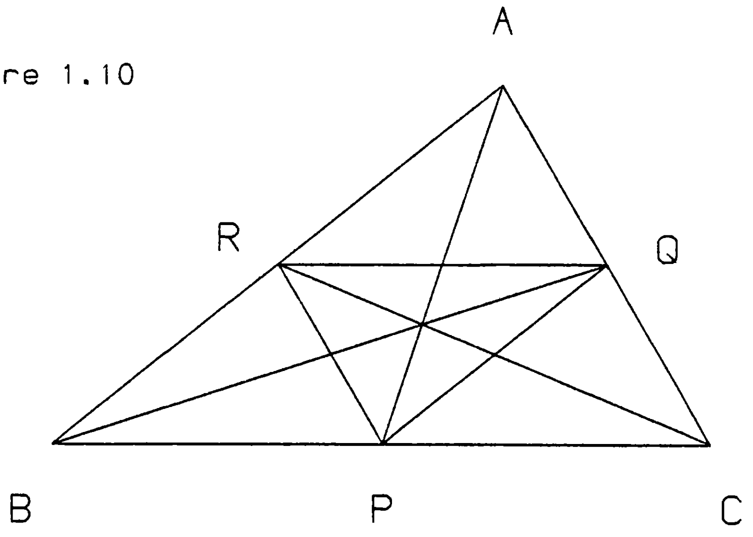
Figure 1.9

triangle, and P, Q and R are located at arbitrary points on each side) provides a unique $C^1$ piecewise quadratic to fit value and gradient data at the triangle's vertices. Moreover they showed that, using this construction, continuous differentiability may be preserved within any triangulation - the interior point 0 within each triangle should be chosen so that, if a pair of triangles have a common edge, then the line joining their interior points intersects the common edge between its vertices. This will always happen if, for example, the incentre of each triangle is chosen.

We would in general prefer to choose the circumcentre rather than incentre of each triangle as its internal point, as this would result in lines joining internal points bisecting triangle edges at right angles. However this is not possible in general because the circumcentre falls outside any triangle which is obtuse. This restriction results in a serious loss of accuracy of approximation, which causes Powell and Sabin to consider also the twelve-triangle subdivision illustrated by Figure 1.10.

By imposing the additional condition that the component of the derivative normal to each edge of $\Delta ABC$ must vary linearly (a condition

Figure 1.10



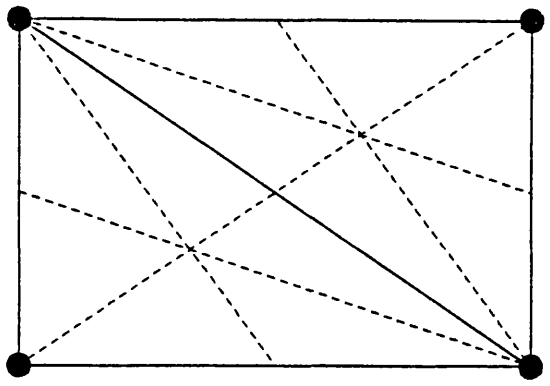
necessary and sufficient for correct reproduction of an arbitrary quadratic), Powell and Sabin succeeded once again in constructing a unique piecewise quadratic surface to fit the (value and gradient) data at the vertices. In this case the construction preserves continuous differentiability across any triangulation with points, P, Q and R chosen as the midpoints of the triangle's sides.

Powell and Sabin's final recommendation was therefore as follows: "In each triangle the points P, Q and R are chosen to be the midpoints of the sides. In each triangle that is sufficiently acute, for instance this may mean that no angle exceeds $75^o$, we define $\phi(x, y)$ by the (6-triangle) method that chooses O to be the circumcenter of $\Delta ABC$. In all other cases we apply the 12-triangle method, where the required normal derivatives are obtained by linear interpolation".

There is undoubtedly considerable merit in the method proposed by Powell and Sabin (though once again we are not presented with any examples of its practical realisation); however it cannot be considered totally satisfactory as a general method for contouring known functions:- in practice it is attractive to be able to use a rectangular (usually square) grid rather than a triangular one. Such a grid may of course be triangulated using right-angled triangular elements, but the problem arises that we are faced with an arbitrary choice as to which diagonal should be used to split a rectangular cell into two triangular ones; and the result of this choice will affect the nature of the approximation. This is a highly unsatisfactory state of affairs and for this reason we now finally consider the problem of constructing rectangular seamed quadratic finite elements to fit value and gradient data on a rectangular grid.

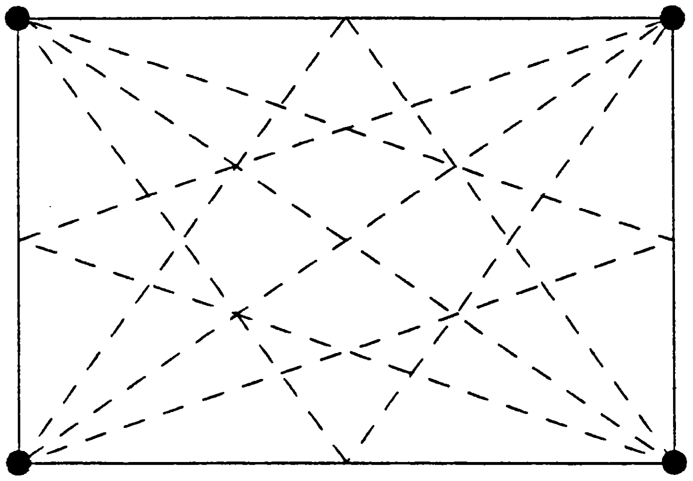
To this author's knowledge the only successful attempt at such a task prior to the work described later in this thesis is due to Lancaster and Ritchie (Ritchie, 1978). Their seamed element is based on a special case of Powell and Sabin's 6-triangle seamed element. Figure 1.11 illustrates a pair of right-angled Powell-Sabin elements arranged in such a way that together they may be used to fit a $C^1$ piecewise quadratic surface to value and gradient data at the vertices of a rectangle. It is easy to demonstrate that if such 'composite' elements are used to

Figure 1.11



approximate a surface across a complete grid, the resulting approximant surface is also continuously differentiable. However, as we have just pointed out, the rectangular element could equally have been constructed from a pair of triangular elements whose common edge was the other diagonal of the rectangle, resulting in a significantly different approximation. Ritchie demonstrates that the difference between these two schemes can sometimes be great, and therefore recommends the compromise of employing the mean of the two possibilities, which results in the seamed quadratic element illustrated by Figure 1.12.

Figure 1.12



The Lancaster-Ritchie element comprises a total of 32 quadratic pieces; however eight of these pieces are quadrilaterals and therefore a further eight internal subdivisions are required to produce a (40-triangle) construction which is suitable for contouring using the routine of Marlow and Powell.

Once again published examples of the practical implementation of this element are lacking, and it is therefore difficult to judge how well it performs. However, given the fact that Powell and Sabin's elements may be used to construct a six-triangle-per-grid-point approximation for data on a triangular grid, the large number of quadratic pieces in the Lancaster-Ritchie element seems rather excessive. At the time when this project was begun it was however the only rectangular seamed piecewise quadratic element known to this author.

## 1.5 Summary of what follows in the thesis

The subject of this thesis is the investigation and exploitation of a new rectangular seamed quadratic finite element comprising just sixteen quadratic pieces: half the number contained in the Lancaster-Ritchie element. The discovery of this new element is due to Professor R. Sibson, but a study of its properties and an implementation of the contouring method of which it forms the basis have been carried out by this author; these are described in the pages which follow.

We begin in Chapter 2 by introducing the element and proving that the use of several such elements juxtaposed across a rectangular grid of value and gradient data defines a unique $C^1$ piecewise quadratic surface suitable for contouring by a routine such as that of Marlow and Powell. Much of the material in this chapter may be found in Sibson and Thomson (1981).

Chapter 3 provides a detailed explanation of how the piecewise quadratic method was implemented in the form of CONICON, a Fortran subroutine package which incorporates a wide range of features including contour annotation and crosshatching between contour levels.

We then proceed in Chapter 4 to apply the CONICON package to the contouring of a collection of data sets arising in a wide variety of disciplines. In some cases we are able to compare the performance of CONICON with other contouring packages, and results are encouraging in all cases.

In Chapter 5 we present an error analysis of the seamed quadratic element, conducted with the aid of CAMAL, a computer package which carries out algebraic manipulations; results are used to derive bounds for the error involved in approximating a known function by the element. We consider ways in which our results might be put to practical use both in the design and in the analysis of contour maps produced by piecewise quadratic approximation.

In Chapter 6 we explore the idea of varying the size of grid cells within a single plot in order to even out local fluctuations in error of approximation (results derived in the previous chapter are put to use here).

A computational implementation is described which is based on the concept of the quad tree, and a number of alternative methods of constructing the grid are investigated. Results here are inconclusive and it is believed that further work in this area might be particularly beneficial.

Finally we conduct a comparison of existing contouring packages, based on user documentation alone, or in some cases only on sales literature. A large and varied selection of packages is discussed, but it is felt that, in terms of quality of output, few can be regarded as serious competitors to CONICON.

# CHAPTER 2

# A SEAMED QUADRATIC ELEMENT FOR CONTOURING

## 2.1 Introduction

In Chapter 1 we discussed the possibility of contouring a smooth function by evaluating its heights and gradients at the nodes of a regular grid, approximating the surface within each grid cell by a piecewise quadratic function, and plotting the contours of the resultant surface using the Marlow-Powell (1976) or some similar quadratic contouring algorithm. We noted that the seamed elements of Powell and Sabin (1977) and Lancaster and Ritchie (Ritchie, 1978) provided suitable approximants over triangular and rectangular grids respectively; however it was felt that a method which used the Lancaster-Ritchie element would be of limited practicability due to the large number (32) of regions which comprise the element. Indeed it would have to be subdivided further, into forty triangles, to be used in conjunction with the Marlow-Powell algorithm.

In this chapter we introduce a sixteen-triangle rectangular element which carries out the same task as the Lancaster-Ritchie element more economically. We therefore propose to use the sixteen-triangle element as the basis of a method for contouring data over a rectangular grid, and we proceed with the development and application of this method in this and subsequent chapters of the thesis.

The element which we introduce is indeed optimally parsimonious, in the sense that the specification of value and gradient data at the vertices, and the requirement of linearity of normal component of the gradient along the edges, leave no spare degrees of freedom. The linearity condition ensures that when such elements are juxtaposed to form a grid, the resultant function is continuously differentiable across the grid lines. On a rectangular grid the use of this element leads to a sixteen-triangles-per-grid-point subdivision, a performance reasonably competitive with the 12-triangle subdivision obtained on a triangular grid using the Powell-Sabin element.

The subdivision employed is shown in Figure 2.1. It consists (to borrow Professor Lancaster's terminology) of four rectangular subelements arranged in a St. George pattern, each being internally subdivided in a St. Andrew pattern.

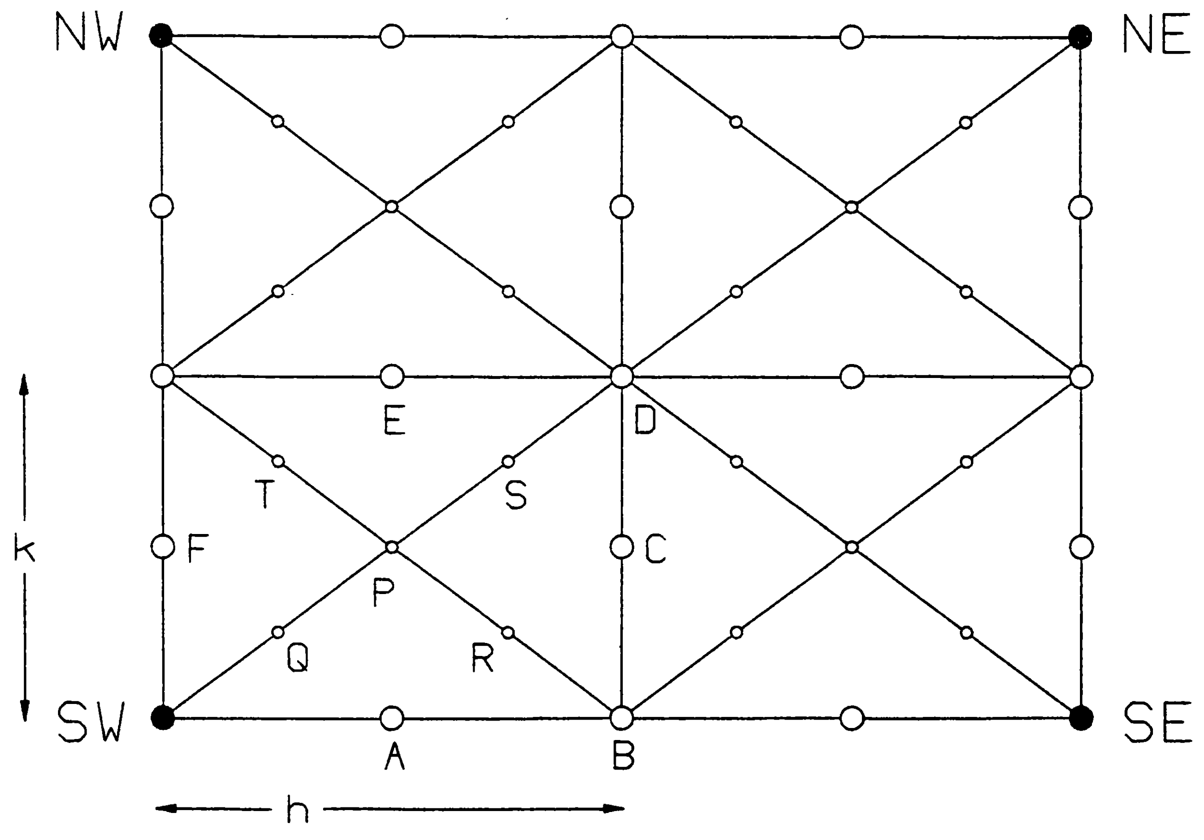


Figure 2.1

Figure 2.1 shows some of the notations and conventions we use: the element is 2h x 2k, its four corners, at which value and gradient are given, being SW, SE, NE, NW. The value and eastwards and northwards components of the gradient at a point are written (z; zx, zy).

## 2.2 A quadratic element on the line

We first establish a result for a 1-dimensional piecewise quadratic element. The result is a consequence of the tangent intersection property of quadratic functions, which may be stated as follows: given an arbitrary quadratic Q on the line, with tangents constructed at any two points, say $(x_1, Q(x_1))$ and $(x_2, Q(x_2))$, then whatever the coefficients in Q are, the tangents have a common value at $x = \frac{1}{2}(x_1 + x_2)$. Thus we define the tangent intersection value of a quadratic on a line segment to be the value taken by tangents constructed at either end of the quadratic at their point of intersection mid-way along the line segment.

Now consider a line segment of length 2h, divided at its centre (the origin). We show that the specification of value and gradient at the ends ($z_L$, $g_L$ at $-h$, $z_R$, $g_R$ at $+h$) determines without spare degrees of freedom a two-segment continuously differentiable quadratic element, and we obtain a relationship between the tangent intersection values of the two quadratics and the value at the origin. Figure 2.2 explains the notation.

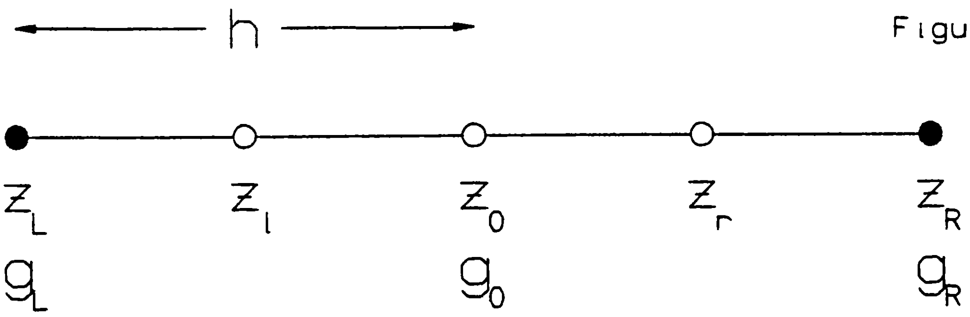


Figure 2.2

We first make a change of parametrisation, replacing $g_L$, $g_R$ by the tangent intersection values $z_L + \frac{1}{2}hg_L$, $z_R - \frac{1}{2}hg_R$ ( $= T_L$, $T_R$). It is then easy to check that the quadratics

$$Q_L(x) = z_0 + g_0x + p_Lx^2 \quad (-h \le x \le 0)$$

$$Q_R(x) = z_0 + g_0x + p_Rx^2 \quad (0 \le x \le h)$$

where

$$z_0 = \tfrac{1}{2}(T_L + T_R) \tag{2.1}$$

$$hg_0 = T_R - T_L \tag{2.2}$$

$$h^2 p_L = z_L - {}^3\!/_2\, T_L + \tfrac{1}{2}\, T_R \tag{2.3}$$

$$h^2 p_R = z_R + \tfrac{1}{2}\, T_L - {}^3\!/_2\, T_R \tag{2.4}$$

uniquely have the desired property. Equation(2.1) is obviously a necessary and sufficient condition on the three values for the two quadratics to have a common gradient at 0. Figure 2.3 gives a pictorial representation.

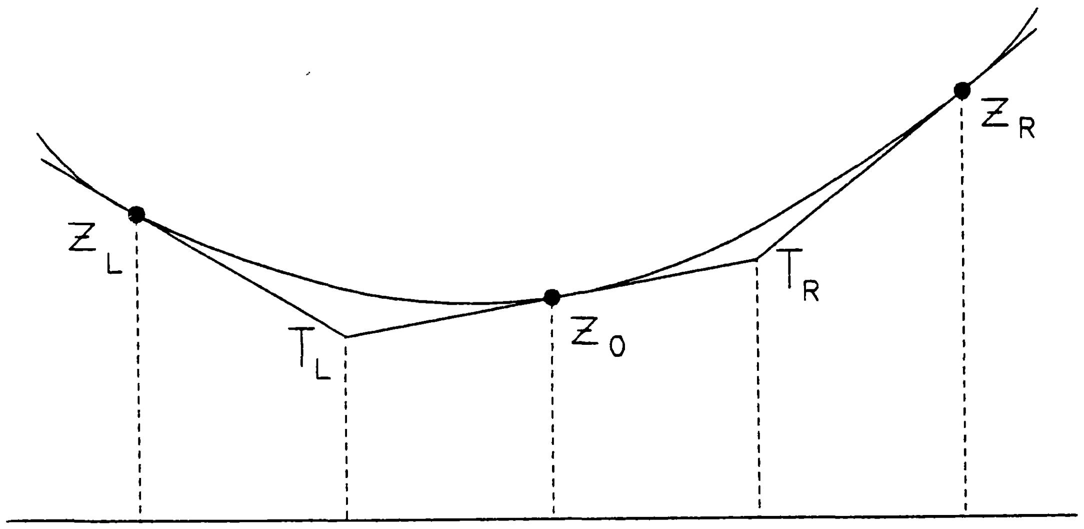


Figure 2.3

## 2.3 Continuity and Uniqueness

Equation (2.1) provides us with a simple method of checking that a $C^1$ surface can be fitted over our element. Beginning with value and gradient data at the vertices of the element, we apply equation (2.1) repeatedly until a comprehensive picture has been built up of the values along the element's seams which are analagous to the values $T_L$, $T_R$ and $z_0$ in the 1-dimensional element described in the previous section. We check for inconsistencies in these values, and on finding none, conclude that a continuously differentiable surface may be fitted over the element as a whole.

We write expressions as hollow arrays, in terms of corner values and tangent intersections, that is

$$\begin{bmatrix} z_{NW} & T_{NW}^{x} & T_{NE}^{x} & z_{NE} \\ T_{NW}^{y} & & & T_{NE}^{y} \\ T_{SW}^{y} & & & T_{SE}^{y} \\ z_{SW} & T_{SW}^{x} & T_{SE}^{x} & z_{SE} \end{bmatrix}$$

where

$$T_{NW}^{x} = z_{NW} + {}^{h}/_{2}\ s_{NW}$$

$$T_{NW}^{y} = z_{NW} - {}^{k}/_{2}\ t_{NW}, \qquad \text{etc.}$$

Thus, on applying (2.1) to $T_{SW}^{x}$ and $T_{SE}^{x}$ we see that the height of the surface at B (Figure 2.1) can be written as

$$\tfrac{1}{2}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & & & 0 \\ 0 & & & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Using the condition of linearity of the normal component of the gradient, it follows that the tangent intersection at C ($T_C$) has value

$$\tfrac{1}{2}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & & & 0 \\ 1 & & & 1 \\ -1 & 1 & 1 & -1 \end{bmatrix}$$

Similarly the tangent intersection on the north side of D will have value

$$\frac{1}{2} \begin{bmatrix} -1 & 1 & 1 & -1 \\ 1 & & & 1 \\ 0 & & & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Application of (2.1) therefore gives the value at D as

$$\frac{1}{4} \begin{bmatrix} -1 & 1 & 1 & -1 \\ 1 & & & 1 \\ 1 & & & 1 \\ -1 & 1 & 1 & -1 \end{bmatrix}$$

From the symmetry of this expression it is immediately obvious that the same result would have been obtained if we had approached D from east and west rather than from north and south. Thus continuous differentiability has been established along the St. George seams, and it only remains to show that the surface is $C^1$ within each St. Andrew subelement.

Now clearly

$T_R = \frac{1}{2} (T_A + T_C)$, $T_T = \frac{1}{2} (T_E + T_F)$

Thus, from (2.1), the value at P is

$\frac{1}{4} (T_A + T_C + T_E + T_F)$

$$= \ {}^1/_8 \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & & & 0 \\ 4 & & & 1 \\ -2 & 4 & 1 & -1 \end{bmatrix}$$

from south east – north west approaches.

Now approaching P from the south west and north east, we find

$T_Q = \frac{1}{2} (T_A + T_F)$, $T_S = \frac{1}{2} (T_C + T_E)$

so again $\frac{1}{4} (T_A + T_C + T_E + T_F)$ is the value at P.

In this way we fill in values in all four subelements, and there are no inconsistencies.

We have now shown that a continuously differentiable surface can be fitted over our element. To show that this surface is uniquely defined we must prove that it is impossible to fit a non-zero surface to data consisting of zero value and gradient at each vertex. Now the one dimensional element described in Section (2.2) is identically zero if the value and gradient at the end points are zero. Thus the surface must vanish along our element's edges. Also, as a result of the gradient restriction which has been imposed, the inwards component of the derivative at the midpoint of each edge is zero; hence the internal south-north and west-east dividing lines are identically zero, so the derivatives at D are zero. We have now shown that each St. Andrew cross diagonal has zero value and gradient at its ends, and is therefore identically zero. Thus all the seams of the element are identically zero, and the surface vanishes over the element as a whole. Uniqueness has been established.

## 2.4 Some expressions required for contouring

As has already been indicated, the major use to which we intend to put the seamed element described above is in contouring, using a subroutine such as that of Marlow and Powell (MP) to trace the conic sections over each triangle. MP exploit the fact that a quadratic on a triangle may conveniently be parametrised by its values at the vertices and the midpoints of the edges, and these data are required as input for their routine. We therefore give explicitly the values at A, B, C, D, P, Q, R, S (see Figure 2.1) as linear combinations of values and normalised inwards components of derivatives that is

$$\begin{bmatrix} z_{NW} & hs_{NW} & -hs_{NE} & z_{NE} \\ -kt_{NW} & & & -kt_{NE} \\ kt_{SW} & & & kt_{SE} \\ z_{SW} & hs_{SW} & -hs_{SE} & z_{SE} \end{bmatrix}$$

All other values follow by symmetry. The linear combinations are as follows:-

$$A:\quad {}^{1}/_{16}\begin{bmatrix}0&0&0&0\\0&&&0\\0&&&0\\14&5&1&2\end{bmatrix}\qquad B:\quad \tfrac{1}{4}\begin{bmatrix}0&0&0&0\\0&&&0\\0&&&0\\2&1&1&2\end{bmatrix}$$

$$C:\quad {}^{1}/_{32}\begin{bmatrix}2&1&1&2\\1&&&1\\5&&&5\\14&7&7&14\end{bmatrix}\qquad D:\quad {}^{1}/_{8}\begin{bmatrix}2&1&1&2\\1&&&1\\1&&&1\\2&1&1&2\end{bmatrix}$$

$$P:\quad {}^{1}/_{16}\begin{bmatrix}2&1&0&0\\1&&&0\\4&&&1\\12&4&1&2\end{bmatrix}\qquad Q:\quad {}^{1}/_{64}\begin{bmatrix}2&1&0&0\\1&&&0\\12&&&1\\60&12&1&2\end{bmatrix}$$

$$R:\quad {}^{1}/_{64}\begin{bmatrix}2&1&0&0\\1&&&0\\8&&&5\\44&20&9&18\end{bmatrix}\qquad S:\quad {}^{1}/_{64}\begin{bmatrix}14&7&2&4\\7&&&2\\14&&&7\\32&14&7&14\end{bmatrix}\tag{2.5}$$

We note that all coefficients encountered in these linear combinations are binary fractions with largest denominator 64. This makes the actual computation as numerically stable as we could wish.

It is not necessary, however, to use the linear combinations above directly in evaluating the heights at P, Q, R and S. Instead, we may find it more convenient to use the expressions above only to calculate values along the external boundaries of the element and on the St. George seams. Now it can be shown (see Powell (1974)) that the eight peripheral values on each rectangular subelement uniquely define the surface within that subelement. Thus we can write those values internal to the subelement as linear combinations of the peripheral values. The following expressions are used to evaluate P and Q. Again all other values follow by symmetry.

$$P:\ \tfrac{1}{4}\begin{bmatrix} -1 & 2 & -1 \\ 2 & & 2 \\ -1 & 2 & -1 \end{bmatrix} \qquad Q:\ {}^{1}/_{16}\begin{bmatrix} -3 & 2 & -1 \\ 10 & & 2 \\ -1 & 10 & -3 \end{bmatrix}$$

In practice, however, a slightly different approach turns out to be faster still. Examination of the coding of the MP routine reveals that the algorithm uses the function values at the midpoints of the triangle's sides once only, to evaluate the tangent intersection values at those points. We may therefore reduce the amount of work done by the MP routine a little by passing tangent intersection values to it directly, in place of the function values themselves. Again we find it convenient to carry out a two-stage process:- first we evaluate height and tangent intersection data along the boundary of the element, plus the height at the centre and tangent intersection values on the St. George seams (an expression for the tangent intersection value at C is given in the previous section). We can then evaluate the five values inside each subsquare as linear combinations of the four tangent intersection values on the borders of that subsquare. Each internal tangent intersection is simply the mean value of the two nearest tangent intersections, and the central value is the mean of all four peripheral tangent intersections. Thus the calculations required are even simpler than those given by the expressions (2.5) and (2.6).

Finally, for completeness, we give expressions for the tangent intersection values at A, C, Q, R, S as direct linear combinations of values and normalised inwards components of derivatives. They are as follows:-

$$T_A:\ \tfrac{1}{2}\begin{bmatrix}0&0&0&0\\0&&&0\\0&&&0\\2&1&0&0\end{bmatrix}\qquad T_C\ \tfrac{1}{4}\begin{bmatrix}0&0&0&0\\0&&&0\\1&&&1\\2&1&1&2\end{bmatrix}$$

$$T_Q:\ \tfrac{1}{4}\begin{bmatrix}0&0&0&0\\0&&&0\\1&&&0\\4&1&0&0\end{bmatrix}\qquad T_R:\ {}^{1}/_{8}\begin{bmatrix}0&0&0&0\\0&&&0\\1&&&1\\6&3&1&2\end{bmatrix}$$

$$T_S:\ {}^{1}/_{8}\begin{bmatrix}2&1&0&0\\1&&&0\\2&&&1\\4&2&1&2\end{bmatrix}\tag{2.7}$$

## 2.5 Application of the method: an introduction

Before plotting it is usually convenient to link the partial contours produced by the quadratic contouring routine into complete contours, which will normally either be closed loops or will begin and end on the boundaries of the grid. This makes it possible to reduce 'pen-up' movement by the graphics device - a significant consideration with a pen plotter - and also allows for such refinements as contour annotation and properly constructed broken-line contours. Figure 2.4 shows a single seamed element in which such a linking process has been carried out; the data are shown at the vertices, and the element has been contoured with annotation.

The power of the method which we are proposing should be immediately apparent from this first illustration. The acceptability of any contour as actually drawn by a normal graphics device is measured by two criteria; the extent to which it truly represents the function being contoured and also its visual smoothness. In this example there is strong evidence to

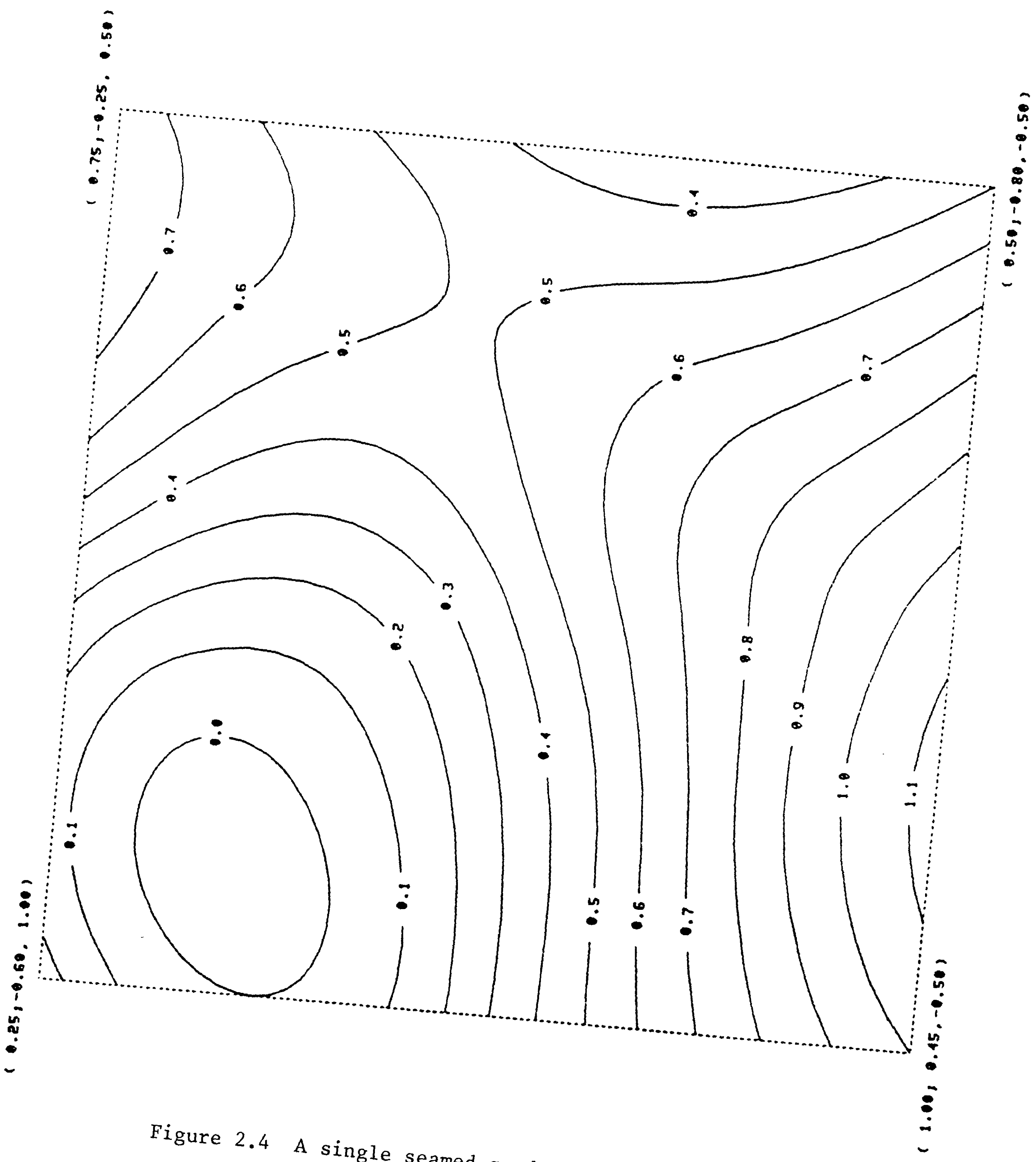


Figure 2.4  A single seamed quadratic element.

suggest that the method can produce contours satisfying both criteria without having to use a particularly fine grid; whereas piecewise linear methods have only the grid size as a control parameter, the use of a continuously differentiable piecewise quadratic approximant allows us to control approximation accuracy and visual quality independently:- the fineness of the grid controls the former and the typical segment length controls the latter. Moreover, accurate approximation can be obtained without using a particularly fine grid; since the approximant is fitted using value and gradient rather than just value, accurate approximation is possible at much coarser grid spacings than with the piecewise linear approach:- in this example both a local minimum and a saddle point (the feature which piecewise linear methods find most troublesome) have been accommodated into a single element. MP's technique varies the length of individual line segments according to the curvature of the contour, and it is quite easy to produce contours of excellent visual quality. Of course we cannot reproduce second derivative continuity in the function which we are contouring, but this is of limited importance since the eye is very bad at detecting discontinuities in curvature - it is much more skilled at detecting discontinuities in gradient.

Figure 2.5 shows a complete contour map produced by the method. This plot may be compared with Figure 1.6, which shows the same function (1.6) contoured by a piecewise linear method. The grid in this example is the same as that used to produce Figure 1.6, that is a 30 x 20 grid of elements or a 31 x 21 grid of points. It is felt that comparisons between the two plots demonstrate very convincingly the superiority of the piecewise quadratic method over the piecewise linear method.

Of course, given equal grid sizes, the piecewise linear method is much faster than the piecewise quadratic method (in this example it is approximately three times faster); however to achieve the degree of smoothness (and accuracy) attained in Figure 2.5 using a piecewise linear method

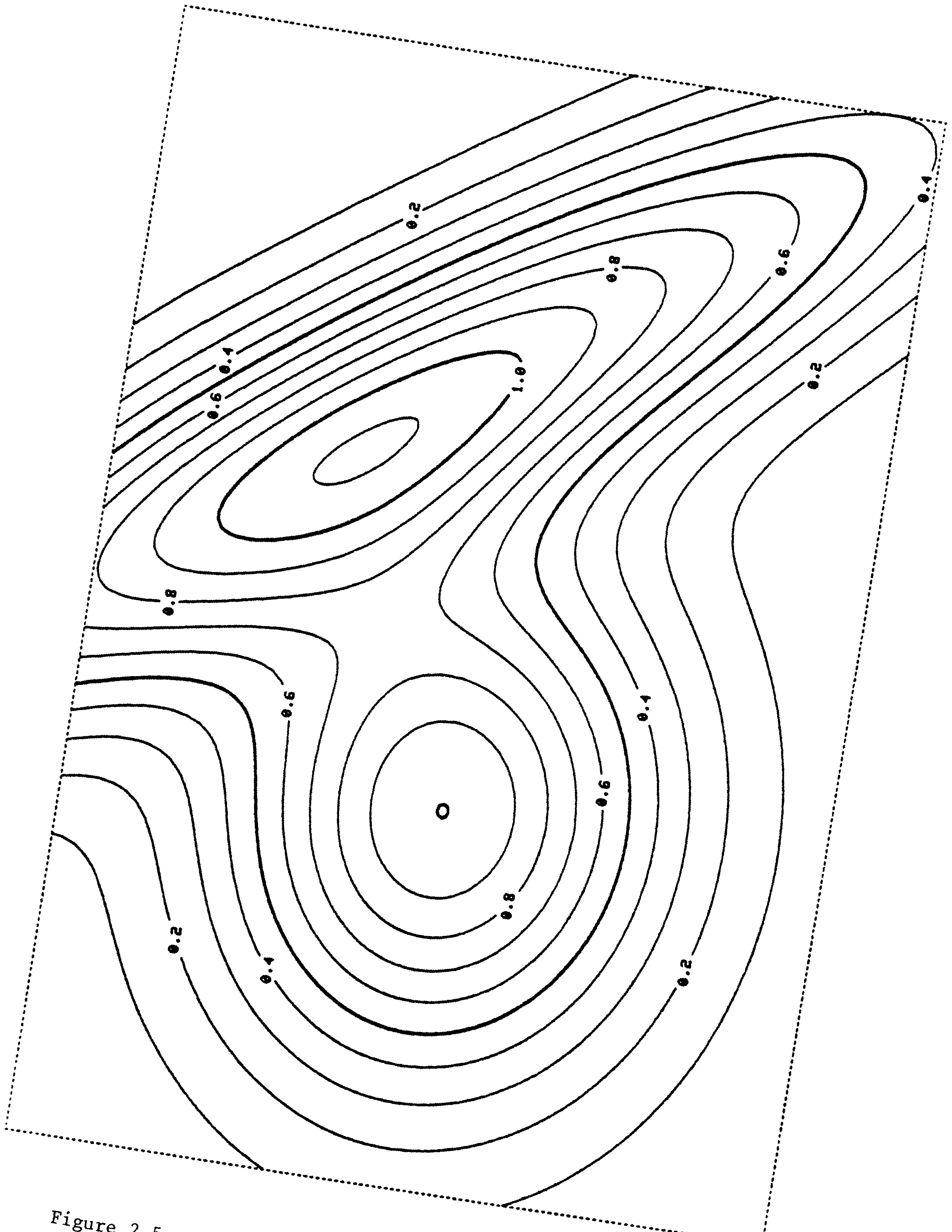


Figure 2.5 Standard function contoured using the seamed quadratic element (30 x 20 grid of elements).

it would be necessary to reduce the grid spacing to such an extent that the method would become prohibitively expensive: in fact reduction to a 120 x 80 grid of elements is still not sufficient to eradicate all the 'corners' in the most sharply curving areas of the contours. Moreover quite acceptable piecewise quadratic plots of this function can be achieved at a lower cost than Figure 2.5:- if the grid size is doubled and a 15 x 10 grid of elements is used (see Figure 2.6) then the plot produced by the piecewise quadratic method is only slightly inferior to Figure 2.5 and takes just under half the time to produce. The only noticeable differences occur in the top contours of the higher peak, and these are only slight.

A single example is not, of course, sufficient information on which to judge the merits of a contouring method; however we do not present any more complete plots at this stage, but instead we refer the reader to the numerous other plots of real and artificial data which appear later in this thesis. It is hoped that the aggregate of these illustrations will be sufficient to convince the reader of the excellent quality achievable with the seamed quadratic element.

## 2.6 Bounds over triangles, etc.

When a large number of elements is used in the production of a contour map, considerable savings in efficiency can be made if good bounds are available for bracketing the range of values which the piecewise quadratic approximant takes in individual triangles, over rectangular subelements and over complete elements. We therefore consider some possibilities here.

In the case of a single triangle, it is not too difficult to calculate exact bounds for the quadratic over its area. If the quadratic is positive or negative definite with its centre $(X_o, Y_o)$ inside the triangle (this is determined at an early stage of the MP algorithm) then one extremum will be the value of the quadratic at $(X_o, Y_o)$ and the other will occur at one

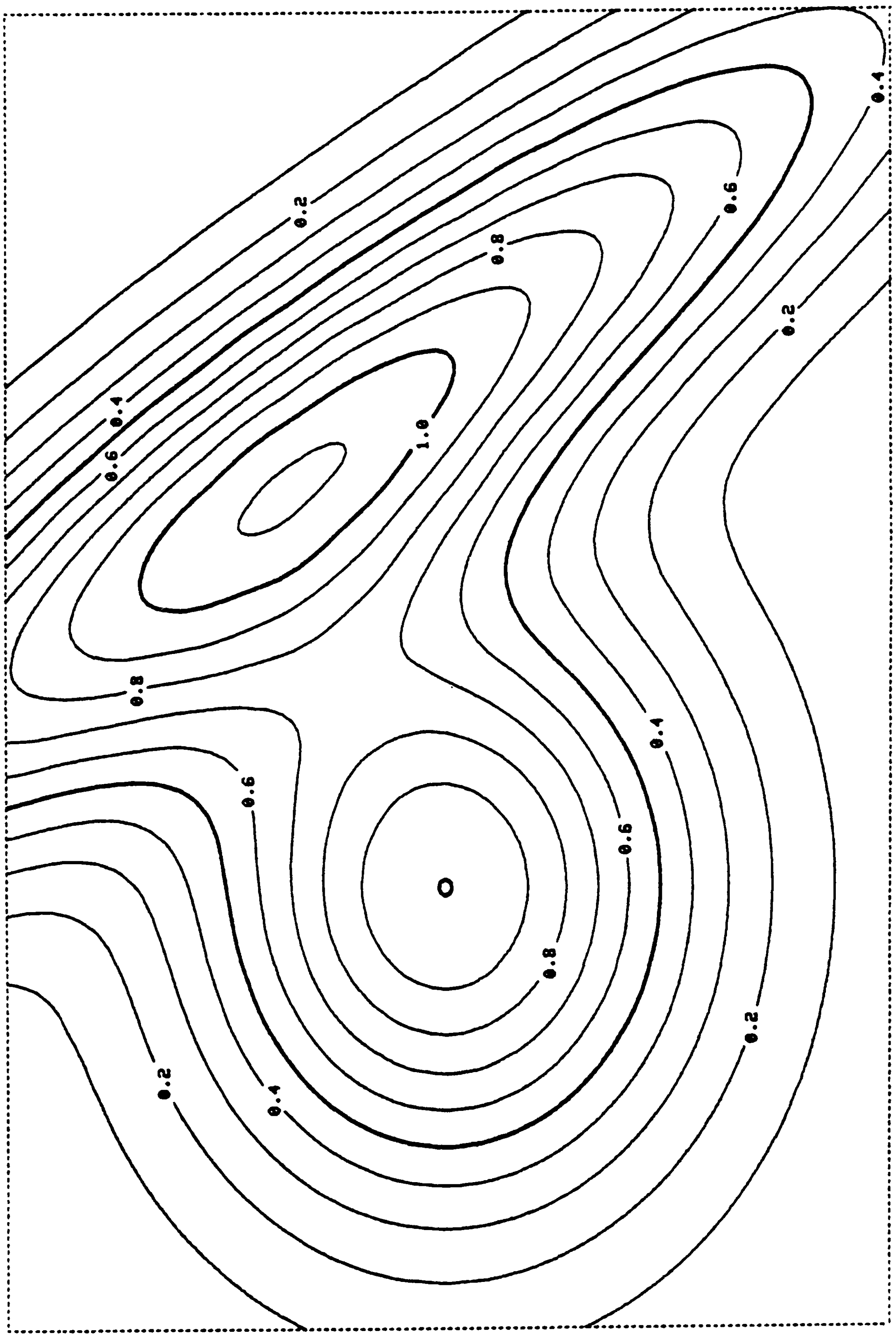


Figure 2.6 Standard function contoured using the seamed quadratic element (15 x 10 grid of elements).

of the vertices. Otherwise the maximum and minimum values will both occur on the triangle's edges. Consider a single edge of the triangle; the maximum and minimum values along that edge will occur at the ends, unless the tangent intersection T on that edge is either larger or smaller than both values $z_L$ and $z_R$ at the ends. In this case one bound for the edge will be

$$(z_L\, z_R - T^2) \;/\; (z_L + z_R - 2T) \qquad (2.8)$$

and the other will be $z_L$ or $z_R$. Repetition of this process on the other two edges yields a set of values whose maximum and minimum are exact bounds for the values taken by the quadratic within the triangle.

Computation of exact bounds therefore takes a non-trivial (though not substantial) amount of time. What we really require is a very fast test, powerful enough to discard a large proportion of those triangles which the current contour does not traverse. As a preliminary step towards deriving such a test, it is convenient at this stage to give a brief introduction to homogeneous (or 'areal') coordinates (Milne, 1924).

Figure 2.7 shows an arbitrary point 0 lying inside a triangle ABC. The

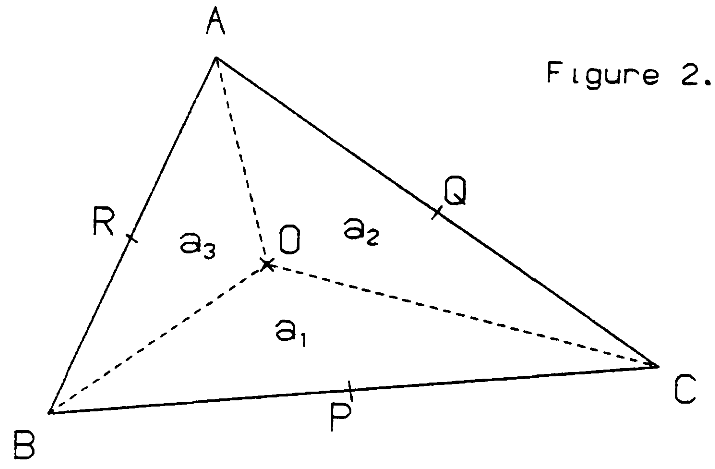


Figure 2.7

areas of triangles BOC, AOC and AOB are $a_1$ $a_2$, $a_3$ respectively. We define the homogeneous coordinates $(x_1, x_2, x_3)$ of 0 with reference to triangle ABC to be $x_1 = \dfrac{a_1}{a_1 + a_2 + a_3}$, $x_2 = \dfrac{a_2}{a_1 + a_2 + a_3}$, $x_3 = \dfrac{a_3}{a_1 + a_2 + a_3}$ . The term in the denominators is not strictly an essential part of the definition (a more general definition would allow any non-zero multiple of $(x_1, x_2, x_3)$), but the normalizing condition $x_1 + x_2 + x_3 = 1$ is convenient for our present purpose. The definition can also be extended to include points outside triangle ABC, but we shall not be interested in such points; we therefore have the additional property that $\underline{x} \geq \underline{0}$, where $\underline{x}^T = (x_1, x_2, x_3)$.

To convert the homogeneous point $(x_1, x_2, x_3)$ to Cartesian coordinates we use the formulae

$$\left.\begin{aligned} X &= x_1 X_A + x_2 X_B + x_3 X_C \\ Y &= x_1 Y_A + x_2 Y_B + x_3 Y_C \end{aligned}\right\} \qquad (2.9)$$

where $(X_A, Y_A)$, $(X_B, Y_B)$ and $(X_C, Y_C)$ are the Cartesian coordinates of the vertices of $\Delta ABC$.

Now if P, Q, R are the mid-points of the sides of $\Delta ABC$ (see Figure 2.7) and the quadratic takes values $f_A$, $f_B$, $f_C$, $f_P$, $f_Q$, $f_R$ at the vertices and mid-points of the edges of $\Delta ABC$, then it is easy to show that the equation of the quadratic in homogeneous coordinates is

$$f(\underline{x}) = \underline{x}^T M \underline{x}$$

where M is a 3 x 3 symmetric matrix $(M_{ij})$ with

$$\left.\begin{aligned} M_{11} &= f_A \\ M_{22} &= f_B \\ M_{33} &= f_C \\ M_{12} &= 2f_R - \tfrac{1}{2}(f_A + f_B) \\ M_{13} &= 2f_Q - \tfrac{1}{2}(f_A + f_C) \\ M_{23} &= 2f_P - \tfrac{1}{2}(f_B + f_C) \end{aligned}\right\} \qquad (2.10)$$

The quantities $M_{12}$, $M_{13}$ and $M_{23}$ are simply the tangent intersection values at R, Q and P respectively.

We are now in a position to consider some easy-to-calculate bounds for the values taken by the quadratic within a single triangle. Some useful ones are given in Lemma 2.1.

Lemma 2.1

A quadratic parametrised by the values which it takes at the vertices and midpoints of the sides of a triangle is bounded above and below within that triangle by the maximum and minimum of the set of values comprising the values at the vertices plus the tangent intersection value on each side.

Proof

Consider the maximum value over the triangle:-

$$\max\{\underline{x}^T M \underline{x} : \underline{x} \geq 0, \underline{1}^T\underline{x} = 1\}$$

$$\leq \max\{\underline{x}^T M \underline{y} : x \geq 0, \underline{1}^T\underline{x} = 1, y \geq 0, \underline{1}^T\underline{y} = 1\}$$

$$= \max_{i,j} M_{ij}$$

Similarly $\min\{\underline{x}^T M \underline{x} : x \geq 0, \underline{1}^T\underline{x} = 1\}$

$$\geq \min_{i,j} M_{ij}$$

and the proof is complete.

The bounds given by this lemma are simple to compute and although they are non-minimal, empirical tests have shown that they are exact bounds in a high proportion (about two-thirds) of cases; typically 98-99% of those triangles which remain after use of these bounds actually contain segments of contour at the current level. These bounds are also useful in enabling us to find bounds for the values taken by the piecewise quadratic over large areas. We now consider bounds for a rectangular subelement (comprising four triangles).

Lemma 2.1 tells us that the values (crosses) and tangent intersections (rings) indicated in Figure 2.8(a) give bounds for the values taken by the

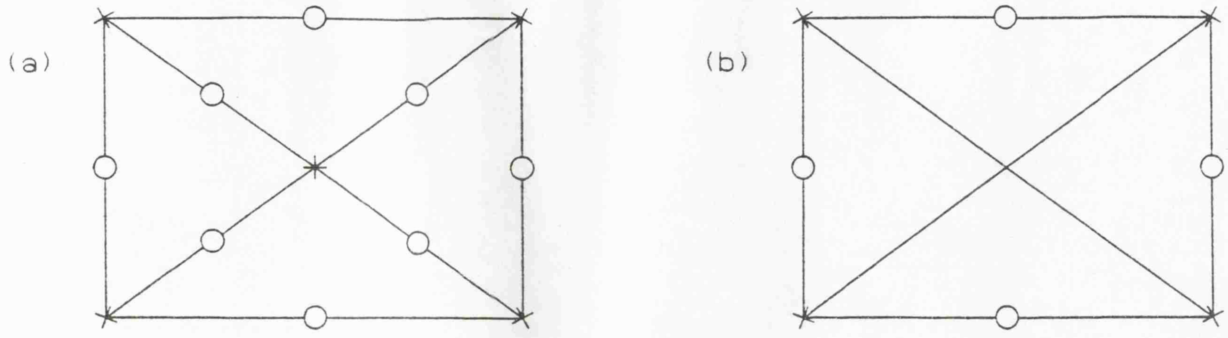


Figure 2.8

piecewise quadratic within the subelement. However, as explained in Section 2.4, the five interior values in this diagram are all weighted averages of the four exterior tangent intersections, so we need not consider the data internal to the subsquare. Thus bounds are given by the eight peripheral values indicated in Figure 2.8(b).

Finally we consider bounds over a complete element. We may obviously begin by taking the aggregate of the values just found for the four sub-elements which form the element. However we can discard the values at the midpoints of the sides and at the centre as a consequence of equation (2.1). We are left with the four values and twelve tangent intersections indicated in Figure 2.9.

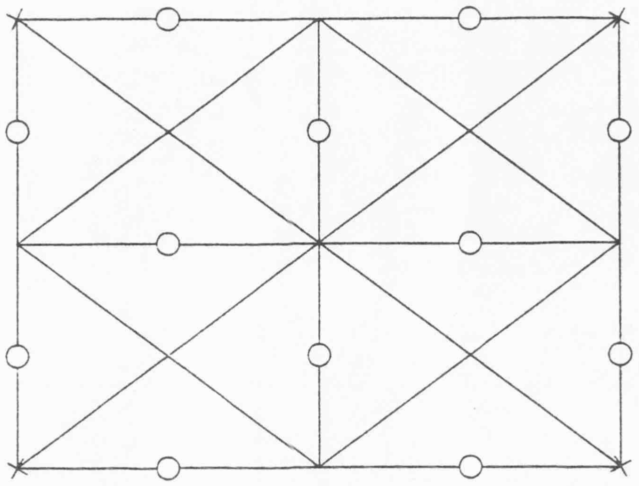


Figure 2.9

In practice, of course, we consider bounds in reverse order to that in which we have just derived them: for each contour level we begin by discarding as many complete elements as the bounds will allow; we then consider the four subelements in each element that remains, discarding those whose bounds do not include the current contour height. For each triangle that remains we begin by making the simple calculation of non-minimal bounds given in Lemma 2.1. If a triangle survives this test we have found that it is so likely that the contour passes across the current triangle that it is inefficient to calculate exact bounds within the triangle; instead we proceed immediately with the quadratic contouring routines.

It is important that the calculation of the bounds discussed above is a numerically stable operation; otherwise there is a danger that areas containing sections of the current contour may be discarded erroneously. Consider the bounds given by Lemma 2.1, which are simply a subset of the values at the vertices and the tangent intersections on the edges of a triangle. In the context of numerical stability, the calculation of the former is the more crucial of the two operations: the bounds supplied by Lemma 2.1 can only be attained by the function if they occur at the vertices and it has already been stated that they are attained on a large proportion of occasions in practice. In fact it was shown in section 2.4 that the evaluation of both values and tangent intersections on each triangle is a very stable operation. Thus the bounds for a triangle given by Lemma 2.1 are stable, and it follows that the bounds derived for subelements and complete elements will also be reliable.

Of course the phenomenon of rounding error can never be eliminated completely; however it is unlikely to be harmful if it is kept to a minimum and if steps are taken to ensure that when it does occur, it occurs consistently. For example, when implementing the method, it is desirable (leaving aside considerations of efficiency) to calculate all heights and tangent intersection values internal to an element once only; and the data along the edge of an element, which usually have to be evaluated twice, should be calculated in exactly the same way on both occasions, in the sense that identical operations are carried out in identical order. In this way any rounding errors which occur will occur consistently; therefore continuity of the contours can be preserved and differences between the true contour and the plotted one will nearly always be imperceptible.

## 2.7 Breakdown of the Implicit Function Theorem

The existence of a well-defined smooth contour line is guaranteed by the Implicit Function Theorem (see section 1.1); this theorem breaks down in regions where the function is locally constant, or at any other points where it has zero partial derivatives, and the computation must follow the mathematics in failing to produce proper contours at the corresponding levels. The cases of local maxima and minima and saddle points present few difficulties, but areas where the function is locally constant can cause problems. In practice, contours will start to display anomalous behaviour as such levels are approached. The particular form which this behaviour takes in our case is a tendency for such contours to follow the seam lines in elements. Figure 2.10 is an artificial example to illustrate this, suggested by Dr. M.A. Sabin.

On a 2 x 2 grid of unit size elements, zero value and gradient are imposed at all eight peripheral grid points, and the value and gradient at the centre are (1; 4,-4). The unlabelled contours are at $\pm$ 0.0001; they coalesce visually into a close approximation to the non-anomalous part of the zero contour internal to the large square, but follow seam lines closely in an octagonal shape round the edge. This effect carries over in less extreme form to other contours at low positive and negative levels. Figure 2.11 is a practical example where this sort of effect is visible.

The function is a nonnegative probability density estimate (constructed by Dr. B.W. Silverman to investigate metallurgical data collected

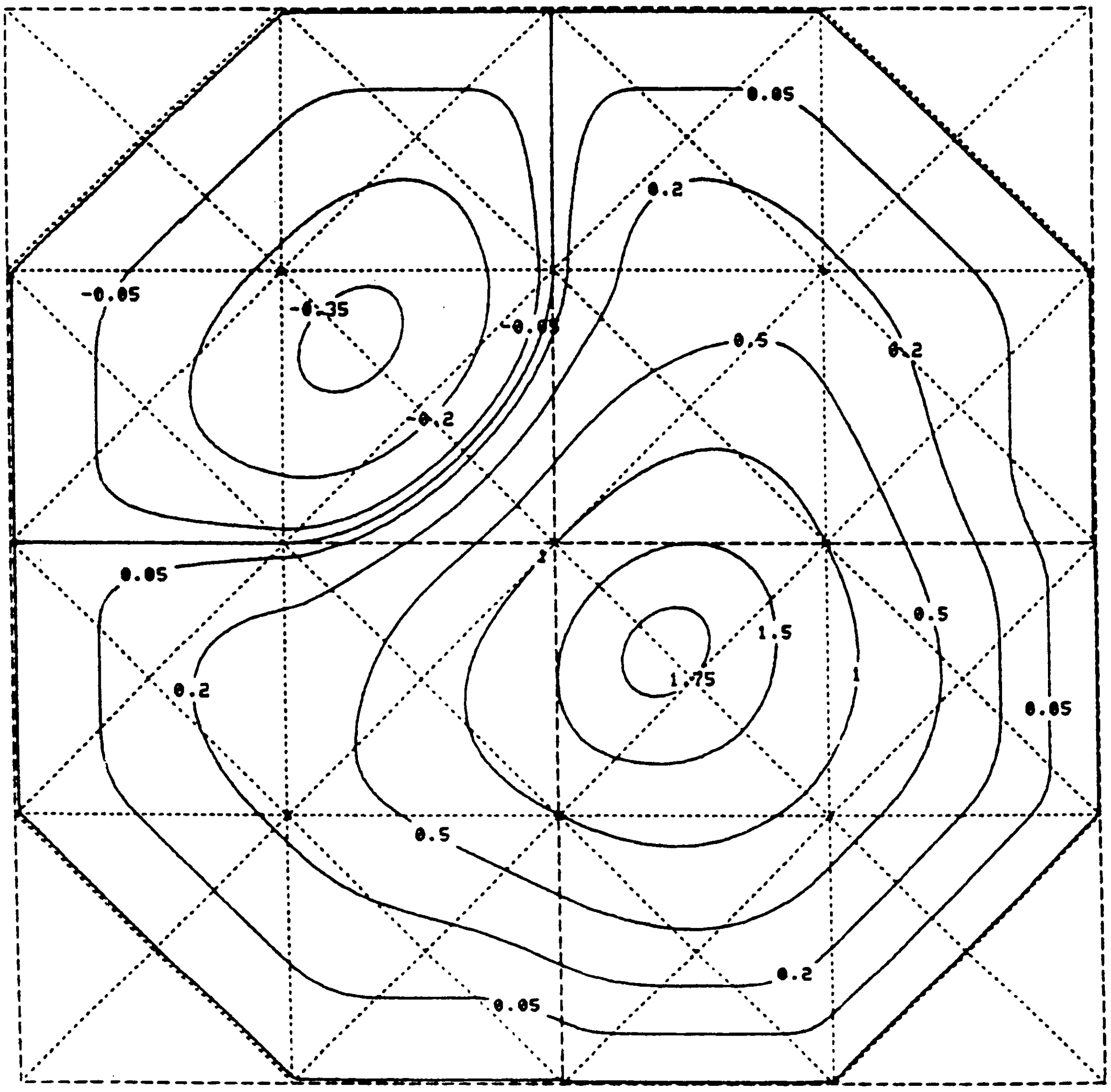


Figure 2.10 Breakdown of the Implicit Function Theorem: an artificial example.

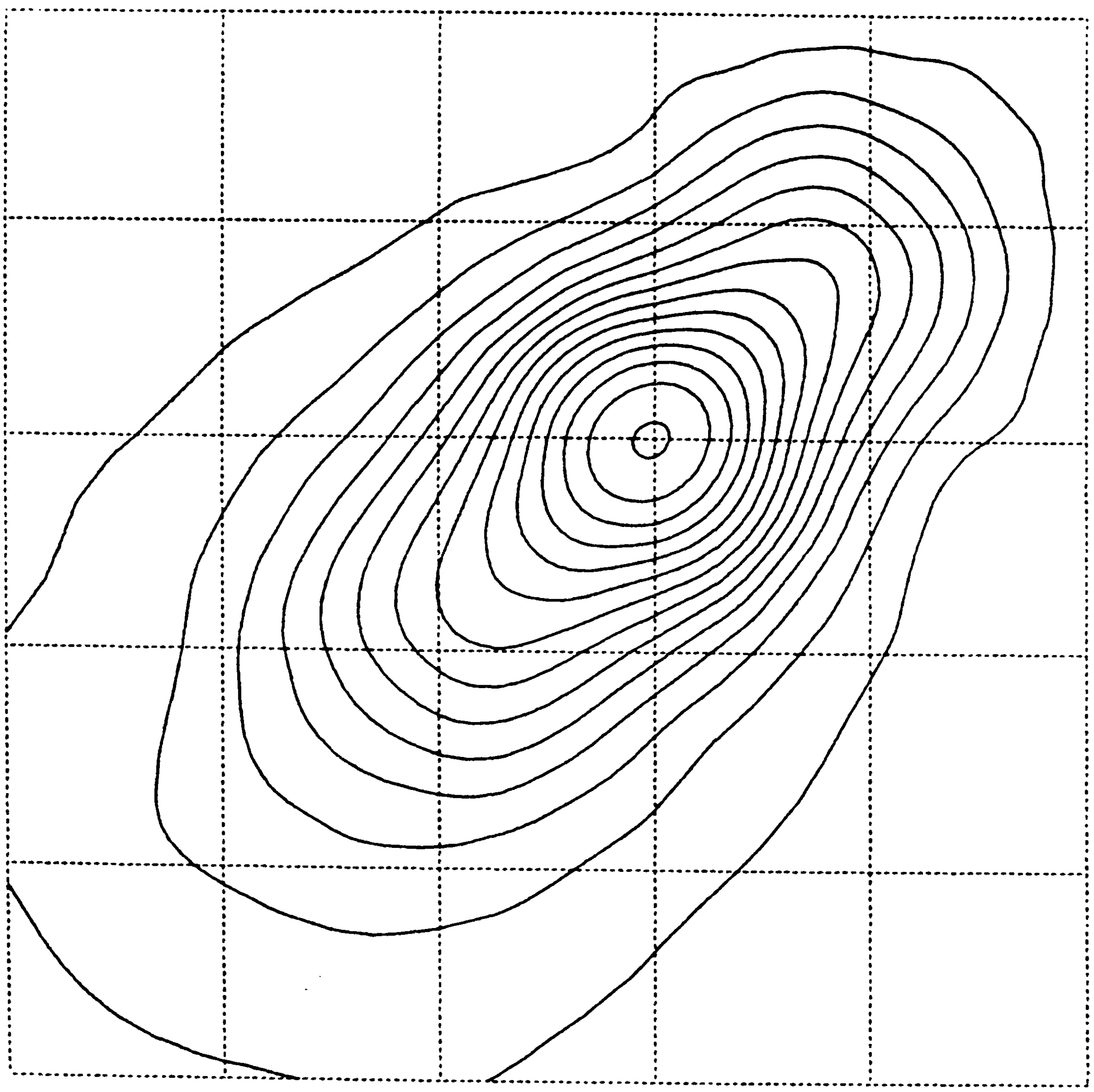


Figure 2.11 Near breakdown of the Implicit Function Theorem in a bivariate probability density estimate.

by Dr. A. Bowyer) which approaches zero closely away from its mode. The oscillations visible clearly in the lowest level contour appear to be associated with the position and size of the 5 x 5 grid of elements; it seems very likely that they arise for the reason explained above. Possible methods of eliminating such oscillations (the most obvious being an overall reduction in grid size) are discussed later in this thesis, in Chapters 4, 5 and 6.

The behaviour just described is not the only anomalous behaviour which may occur in contours of the piecewise quadratic approximant when the Implicit Function Theorem breaks down. It has been stated that contours of the same height will only touch or cross each other in exceptional circumstances; the saddle point is clearly one exception, but it is not the only one. Figure 2.12 illustrates an unusual case in which five contours meet at a single point and three meet at another point.

In this example we have data lying on a 3 x 3 square grid of points with points four units apart, and we are plotting the zero contour. Although some rescaling has been carried out to help the reader assimilate the data easily, it should be emphasized that the data were not constructed artificially, but arose in a practical application in which the $C^1$ natural neighbour method was used to interpolate value-only data on a square grid with missing values. It appears that the true surface was locally constant in this area, but that the influence of surrounding data values (only a small part of the complete grid is shown here) caused the method to estimate non-zero gradients in most cases. Figure 2.13 indicates the positions of the triangular panels and numbers the quadratics in the area of interest from 1 to 10. The quadratics are as follows:-

$$q_1 = q_2 = {}^{9}/_{16}\ xy$$

$$q_3 = -{}^{23}/_{32}\ y^2 + {}^{9}/_{16}\ xy$$

$$q_4 = -{}^{23}/_{32}\ x^2 + {}^{9}/_{16}\ xy$$

$$q_5 = \tfrac{1}{2}\ x^2 - {}^{7}/_{32}\ y^2 - {}^{7}/_{16}\ xy$$

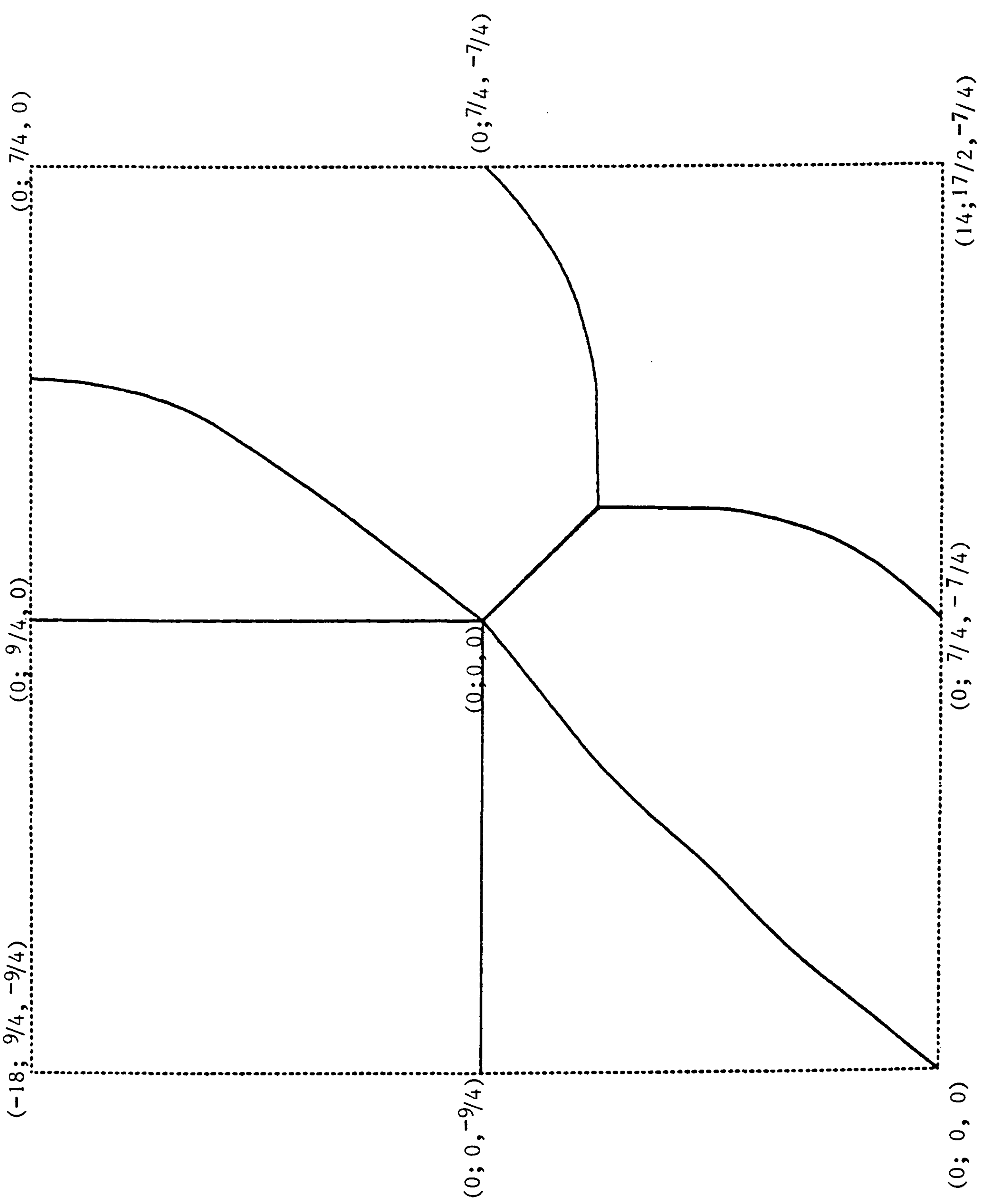


Figure 2.12. Anomalous behaviour caused by breakdown of the Implicit Function Theorem.

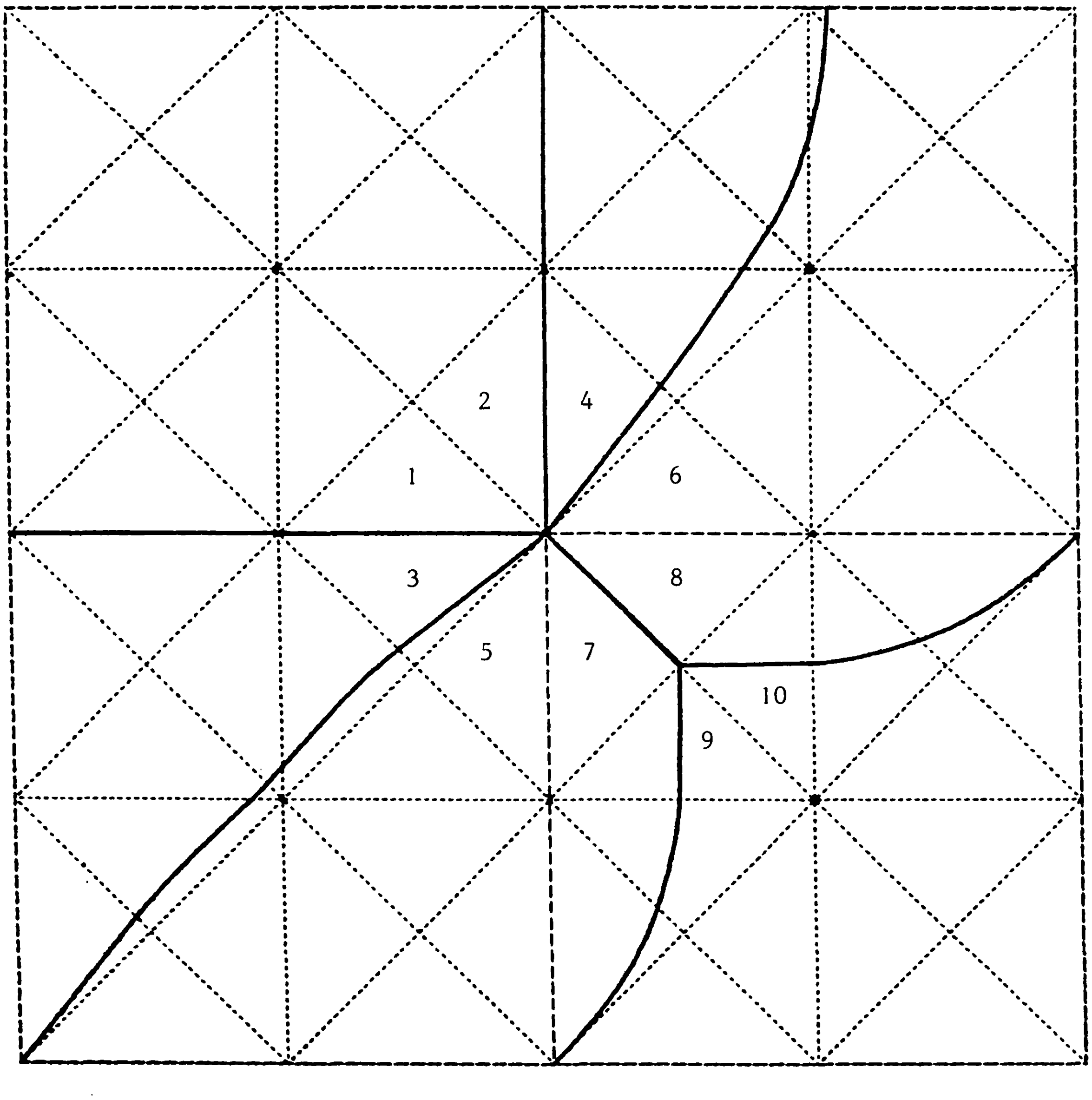


Figure 2.13 Triangular panels plotted over the contour map of Figure 2.12.

$q_6 = -{}^7/32\ x^2 + \frac{1}{2}\ y^2 - {}^7/16\ xy$

$q_7 = q_8 = -{}^7/32\ (x + y)^2$

$q_9 = q_{10} = {}^1/64\ (x^2 + y^2) - {}^{29}/32\ xy + {}^{15}/16\ (-x + y + 1)$

It is easy to verify that these quadratics do indeed form a continuously differentiable surface; we can check that all pairs of neighbouring quadratics fit together smoothly using the following condition:-

Suppose the quadratics $q_i$ and $q_j$ lie on either side of the line

$lx + my + n = 0$

Then the resulting surface is continuously differentiable along the line if and only if

$$q_j\ (x, y) = q_i\ (x, y) + \lambda(lx + my + n)^2 \qquad (2.11)$$

for some constant $\lambda$.

It can now be seen that the approximant is nowhere locally constant, but that the Implicit Function Theorem is broken along a straight line from the origin (the centre of the plot) to the point (1, -1); the contour which occurs here traces the maximum value of the paraboloid $-{}^7/32\ (x + y)^2$. This contour can be considered to be the limiting case, as the contour level approaches zero from below, of pairs of parallel straight-line contours equidistant from the line $x + y = 0$. Thus we can regard the contour as a pair of coincident contours, and this explains the phenomenon of an odd number of contours meeting at a point.

It might at first seem surprising that an example in which five or six contours meet at a point in this manner can occur when the functions being contoured are only polynomials of degree 2; however this is the result of the piecewise nature of the surface and the discontinuities in second derivative which result from this. Consider plotting the contours of linear functions; the contours of a single linear function are simply straight lines, but by piecing together linear functions with discontinuities in first derivative across the joins it is easy to construct a function with an arbitrary number of contours meeting at a single point. This can also be done by piecing together quadratic functions with discontinuities in second derivative where they meet, for example by piecing together sections of hyperbolae around a point in such a way that the asymptotes of each hyperbola run along the boundaries between them.

Examples like the one which we have just discussed cannot be ignored; however it is important to emphasize that they will occur extremely infrequently, and indeed can only occur when our approximant has zero partial derivatives somewhere on the boundary line between two or more quadratics. In general contours produced by our method will be smooth curves of the highest quality.

## 2.8 Comparisons with cubic elements

The 1-dimensional element discussed in Section 2.2 invites comparison with the familiar cubic element. In terms of the conventional approach to error analysis, the latter is certainly to be preferred for the usual applications, but the similarities are remarkably close: equation (2.1), if rewritten as

$$z_0 = \tfrac{1}{2}z_L + \tfrac{1}{4}q_L + \tfrac{1}{2}z_R + \tfrac{1}{4}q_R \qquad (2.12)$$

(where $q_L = hg_L$, $q_R = -hg_R$) holds for both; and both have integral

$$I = h(3z_L + q_L + q_R + 3z_R) / 3 \qquad (2.13)$$

It is not hard to show that there is a piecewise cubic analogue of our sixteen-triangle quadratic element. It is a four-triangle subdivision of the rectangle on the St. Andrew pattern, as illustrated in Figure 2.14, and again is characterised by value and gradient data at the vertices and linearity of the normal component of the derivative along the edges. As in the case of the quadratic element, we use a two-part argument to show

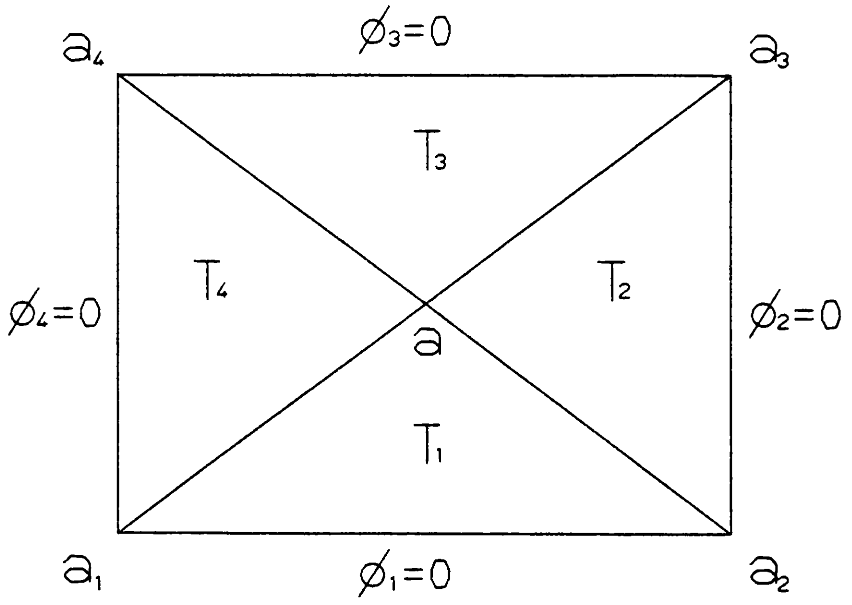


Figure 2.14

that a unique $C^1$ piecewise (cubic) polynomial surface is determined over the element; we show first that it is possible to fit a continuously differentiable surface to arbitrary data and we then prove uniqueness by showing that zero value and gradient data at all vertices must lead to the disappearance of the function throughout the element.

To show that a $C^1$ surface may be fitted to any data set it is sufficient to construct functions (usually termed 'cardinal' functions) satisfying the following two data sets:-

(i) Value = 1 at a single vertex; all other values and gradients equal zero.

(ii) Normalised inwards component of x-derivative = 1 at a single vertex; all other data are zero.

Given such a pair of cardinal functions, the other ten cardinal functions for the element can be constructed trivially using symmetry arguments and it is then possible to fit a $C^1$ surface to any given set of data values by constructing a linear combination of the cardinal functions.

We now explain the notation given in Figure 2.14. The four triangles into which the rectangle R is subdivided are $T_1$, $T_2$, $T_3$, $T_4$, meeting at a, the centre of the element. The vertices of R are labelled $a_1$, $a_2$, $a_3$, $a_4$.

We denote the values of the cardinal function corresponding to value 1 in the south west corner of the element by $\lambda_1$, $\lambda_2$, $\lambda_3$ and $\lambda_4$, the cubic $\lambda_i$ occurring in triangle $T_i$. In the same way we denote the values of the cardinal function corresponding to (normalised) x-derivative 1 in the south west corner by $\rho_{11}$, $\rho_{12}$, $\rho_{13}$ and $\rho_{14}$.

For convenience, but without loss of generality, we regard the centre of the element as the origin and take $h = 1$, $k = 1$. Then the cardinal functions have the following values;-

$$\lambda_1(x, y) = \tfrac{1}{4}x^3 + {}^3/_8\,xy^2 + {}^1/_8\,y^3 + \tfrac{3}{4}xy + {}^3/_8\,(-x - y) + \tfrac{1}{4}$$

$$\lambda_2(x, y) = {}^1/_8\,x^3 - {}^3/_8\,x^2y + \tfrac{3}{4}xy + {}^3/_8\,(-x - y) + \tfrac{1}{4}$$

$$\lambda_3(x, y) = -{}^3/_8\,xy^2 + {}^1/_8\,y^3 + \tfrac{3}{4}xy + {}^3/_8\,(-x - y) + \tfrac{1}{4}$$

$$\lambda_4(x, y) = {}^1/_8\,x^3 + {}^3/_8\,x^2y + \tfrac{1}{4}y^3 + \tfrac{3}{4}xy + {}^3/_8\,(-x - y) + \tfrac{1}{4} \qquad (2.14)$$

$$\rho_{11}(x, y) = \tfrac{1}{4}x^3 + {}^1/_8\,xy^2 - \tfrac{1}{4}x^2 + \tfrac{1}{4}xy - {}^1/_8\,y^2 - {}^1/_8\,x - \tfrac{1}{4}y + {}^1/_8$$

$$\rho_{12}(x, y) = {}^1/_8\,x^3 - \tfrac{1}{4}x^2y - {}^1/_8\,x^2 + \tfrac{1}{2}xy - {}^1/_8\,x - \tfrac{1}{4}y + {}^1/_8$$

$$\rho_{13}(x, y) = -{}^1/_8\,xy^2 + \tfrac{1}{4}xy + {}^1/_8\,y^2 - {}^1/_8\,x - \tfrac{1}{4}y + {}^1/_8$$

$$\rho_{14}(x, y) = {}^1/_8\,x^3 + \tfrac{1}{4}x^2y - {}^1/_8\,x^2 - {}^1/_8\,x - \tfrac{1}{4}y + {}^1/_8 \qquad (2.15)$$

It is straightforward to check that the functions $\lambda_i$ and $\rho_{1i}$ fit the data, that the normal component of the derivative along the edges varies linearly, and that the surface is continuously differentiable across the seams of the element. It remains to be proven that the surface fitted from the obvious linear combination of these and the other cardinal functions is unique. The proof of uniqueness is almost identical to a proof given by Percell (1976) of the uniqueness of the surface determined by the Clough-Tocher triangular element, which was introduced in Clough and Tocher (1965). For use in this proof, we define $\phi_i$ to be the (unique) linear function such that $\phi_i(a) = 1$ and $\phi_i$ vanishes on the exterior edge of $T_i$. Furthermore, let $\phi$ be the function on R defined by

$$\phi|T_i = \phi_i|T_i, \quad 1 \leq i \leq 4$$

where the vertical bar means restriction of a function. $\phi$ is well-defined and continuous on R because $\phi_i$ and $\phi_{i+1}$ are identical on $T_i \cap T_{i+1}$ since both are 0 at $a_{i+1}$ and 1 at a. (Throughout this argument subscripts will be counted mod 4.) We also denote our piecewise cubic by $\gamma$, and the part of it which lies in $T_i$ by $\gamma_i$. So

$$\gamma_i = \gamma|T_i, \; 1 \leq i \leq 4$$

Now when the data at the vertices are zero, $\gamma$ and $\nabla\gamma$ vanish on each exterior edge of T ($\nabla\gamma$ is zero as a result of the gradient restriction). It therefore follows that $\gamma = \ell\phi^2$, where $\ell_i = \ell|T_i$ is a linear function. Note that $\ell$ is continuous because $\gamma$ is continuous and $\phi$ does not vanish on $T_i \cap T_{i+1}$ except at $a_{i+1}$. On $T_i \cap T_{i+1}$, $\nabla\gamma$ is well-defined and is given by either

$$2\ell\phi\nabla\phi_i + \phi^2\nabla\ell_i \qquad \text{or} \qquad 2\ell\phi\nabla\phi_{i+1} + \phi^2\nabla\ell_{i+1}$$

Thus

$$2\ell\nabla(\phi_{i+1} - \phi_i) + \phi\nabla(\ell_{i+1} - \ell_i) = 0$$

on $T_i \cap T_{i+1}$. Since $\phi(a_{i+1}) = 0$ and $\nabla(\phi_{i+1} - \phi_i) \neq 0$ (because the lines $\phi_i = 0$ and $\phi_{i+1} = 0$ are not parallel) it follows that $\ell(a_{i+1}) = 0$. Therefore $\ell_i$ vanishes at the exterior vertices $a_i$, $a_{i+1}$ of the triangle $T_i$, so $\ell_i$ vanishes on the exterior edge of $T_i$. Hence $\ell_i = c_i\phi_i$ for some constant $c_i$. But since $\ell$ is continuous and $\phi_i(a) = 1$, $c_1$, $c_2$, $c_3$ and $c_4$ must be the same, so $\ell = c\phi$ for some real c. Thus $\gamma = c\phi^3$, which cannot be $C^1$ unless $c = 0$ because $\phi$ is not $C^1$. Uniqueness is therefore proven.

The element introduced above and the sixteen-triangle quadratic element preserve the same similarities observed between the two 1-dimensional elements.

For both elements, the value at the centre, $z_0$, is given by the relation

$$z_0 = \tfrac{1}{4}(z_{NE} + z_{NW} + z_{SW} + z_{SE}) + {}^h/8\,(- s_{NE} + s_{NW} + s_{SW} - S_{SE}) +$$

$${}^k/8\,(- t_{NE} - t_{NW} + t_{SW} + t_{SE}) \tag{2.16}$$

and both have integral given by the linear combination

$$\frac{hk}{3}\left[\begin{array}{ccccccc} 3z_{NW} & + & (hs_{NW}) & + & (-hs_{NE}) & + & 3z_{NE} \\ +\ (-kt_{NW}) & & & & & + & (-kt_{NE}) \\ +\ (kt_{SW}) & & & & & + & (kt_{SE}) \\ +\ 3z_{SW} & + & (hs_{SW}) & + & (-hs_{SE}) & + & 3z_{SE} \end{array}\right] \tag{2.17}$$

It is tempting to conjecture that in n dimensions there is a cubic element with $n!2^{n-1}$ simplexes and a quadratic element with $n!2^{2n-1}$ simplexes and that both satisfy the obvious n-dimensional generalisations of (2.12), (2.16) for the value at the centre and of (2.13), (2.17) as integral formula. No attempt has been made to verify this.

If a subroutine were available which employed a method of parametric contour generation for a cubic, analogous to the method used by MP for a quadratic, then it might well be attractive to use the four-triangle cubic element as an alternative to the sixteen-triangle quadratic element; unfortunately no such subroutine has yet been written. It would provide an interesting opportunity to examine the tradeoff between the number of triangles per grid point and the complexity of the function on each triangle. We are therefore forced to confine our attention to the development of the piecewise quadratic method for contouring in the remainder of this thesis.

# CHAPTER 3

# THE CONICON PACKAGE

## 3.1 Introduction

The contouring method proposed in Chapter 2 has been implemented by the author in the form of CONICON, a self-contained package of subroutines which comprises approximately 7500 lines (including copious comments) of ANSI Fortran. Some examples of output from CONICON have already appeared in this thesis; in this chapter we explain the structure of the package, giving details of its more important subroutines, and present further illustrations of its capabilities.

At the highest level of the package are eight master routines, any one of which may be called by the user to produce a complete contour plot with or without features such as annotation, crosshatching and local suppression of contour plotting. A comprehensive description of the functions of these subroutines and the features which they offer can be found in the CONICON users' guide, which forms Appendix A of this thesis. Also documented in Appendix A are a pair of routines for indicating stationary points of the piecewise quadratic approximant, and a number of utility routines which help the user to set up data for his chosen master routine.

At a much lower level, the simple graphics routines which are needed to carry out tasks such as (i) plotting a straight line from the current position to (x, y), or (ii) moving the plotter position invisibly to (x, y), are assumed to be supplied by the user's system and are not an integral part of the CONICON package. The set of such graphics routines which the user is expected to provide is also described in Appendix A.

The CONICON package was developed on the Avon Universities Honeywell Multics system and has been run successfully by the author and a number of other users on several hundred data sets. The package is interfaced to a highly efficient package of graphics subroutines, which allows plots to be displayed directly on both the Tektronix 4015 graphics terminal and the Tektronix 4663 pen plotter belonging to the School of Mathematics at Bath University. Most of the CONICON plots which appear in this thesis are hard copies from the former device, since the latter was not available for most of the duration of this project. Principal advantages of using the latter device are the ability to create larger scale plots and the opportunity to use more than one pen colour in the creation of a single plot (a facility which is utilised by the CONICON package's crosshatching feature).

Following the completion of this project the package was installed (with no major problems) on CDC Cyber 175 and 835 machines at the European Centre for Medium Range Weather Forecasts, Shinfield Park, Reading. A number of further improvements were made by the author to this version of the package, but these were made at too late a date for inclusion in the current chapter and are not documented in Appendix A. However some examples of plots produced by this version of the package appear in Chapter 4. Unlike the other plots in this thesis these were produced by a raster device (Versatec 8122 electrostatic plotter) after vector to raster conversion.

## 3.1.1 Quadratic contouring routines

In the original version of CONICON the routine of Marlow and Powell (1976) was used to trace the contours of the piecewise quadratic approximant across individual triangles. However it soon became apparent that in its published form this subroutine would not be reliable enough for the production of contour maps in large quantities: the routine was found to have an inherent numerical instability, which sometimes resulted in exponent underflow (or overflow) in relatively straightforward non-pathological examples; and it produced incorrect results in some special cases. An attempt to alter MP's routine to improve its stability was not a complete success, and therefore the routine was eventually replaced by a number of new subroutines due to Professor R. Sibson (pers. comm.). The resulting version of the package, CONICON 2, is the one which we describe in this Chapter.

The new routines have a number of distinct advantages over the routine of Marlow and Powell: they have proven to be extremely reliable, are considerably faster (especially over large contour maps) and were carefully designed to ensure that the endpoints of those individual conic sections which in theory are identical are also numerically identical.

Sibson's routines employ a much simpler parametrisation for conic sections which appears a more natural choice than that used by Marlow and Powell and retains a sensible adaptive step length policy. Each conic section is parametrised in terms of its endpoints ((x0, y0, z0) and (x1, y1, z1) in homogeneous coordinates normalised to sum to unity) and the point of intersection (xh, yh, zh) of tangents constructed at the endpoints, which is known as the pole. The normalisation of the pole is dependent on the conic; it is not in general the case that xh + yh + zh = 1. The parametric equations for the conic section are then

$$\begin{aligned} x(\tau) &= x0\tau^2 + 2xh\tau(1-\tau) + x1(1-\tau)^2 \\ y(\tau) &= y0\tau^2 + 2yh\tau(1-\tau) + y1(1-\tau)^2 \\ z(\tau) &= z0\tau^2 + 2zh\tau(1-\tau) + z1(1-\tau)^2 \end{aligned}$$

$$0 \leq \tau \leq 1 \qquad (3.1)$$

These values are unnormalised.

Ring contours lying completely within the reference triangle are divided into two separate pieces, each with its pole at infinity. All cases where the contour degenerates into a pair of straight lines are dealt with correctly by a separate subroutine.

A further advantage of these subroutines is that they always generate conic sections in the same direction, with high ground on the left. At present this fact is ignored by the routine in CONICON which links conic sections to each other, but the potential exists to reduce by 50% the number of comparisons carried out by the linking routine: features such as 'ticking' of contours on the lower (or upper) side could then also be incorporated with no great difficulty.

Experience has indicated that in plots with large numbers of elements the great majority of conic sections need only be approximated by a single straight line; in view of this it is felt that MP's routine with its relatively large overheads wasted a large amount of CPU time making unnecessary calculations. The burden of such calculations imposed by Sibson's routines is light in comparison and this is probably an important contributory factor to the relatively fast speed of these routines. In the current CONICON set-up the number of straight line segments used to approximate a conic section across a triangle is a linear function (rounded to the nearest integer) of 180 degrees minus the angle subtended by the endpoints at the pole, a method which adapts itself according to the fineness of the grid employed and the curvature of the contours. Once this number has been determined the range of the parameter $\tau$ is simply divided into the appropriate number of equal intervals and each interval is represented by a single straight line segment. This rather simple-minded choice of adaptive segment length may well be improveable without the need for the complexity of the MP approach, but it appears to work reasonably well in practice.

## 3.2 Producing a simple contour plot using CONICON

In order to present as simple as possible a description of the basic structure of the CONICON package to begin with, we shall follow through the process of creating a simple contour plot without any of the special features described in the documentation in Appendix A. In addition we shall assume that the user has on file or can calculate both the heights and gradients of the surface which he wishes to contour at the nodes of a square grid. Some of the subroutines encountered in this section will be explained in greater detail below; for the moment we wish simply to summarise the process involved in the creation of a CONICON plot.

The CONICON documentation instructs the user to call either subroutine CONIC1 or CONIC2 to produce any plot without crosshatching, since these are the master routines with the shortest argument lists and are therefore the easiest to use. We shall assume that the user wishes opening and closing of the plot frame to be carried out automatically: therefore he should choose to call subroutine CONIC1. CONIC1 simply opens the plot frame, plots the (rectangular) boundary of the grid in a style chosen by the user, resets the line style to solid for contour plotting, calls subroutine CONIC2 and finally closes the plot frame before returning.

CONIC2 is another short subroutine which sets the dimensions of a number of arrays not required when the crosshatching feature is not being used, and in examples such as ours in which the local contour suppression feature is not wanted sets all values in the array ZLIM to zero (see 3.7 or CONICON documentation for further explanation). Finally it calls subroutine ALLCON before returning.

Subroutine ALLCON is a vital part of the CONICON structure. This routine is called, directly or indirectly, by all master routines in the package with the exception of those which are used to contour the gradient of the approximant surface. In the case of our simple plot this routine will carry out the following tasks:-

(a) Check for illegal values of a number of variables.

(b) Calculate the bounds given in Section 2.6 for values taken by the surface within each element in the grid and store in the array ZLIM.

(c) Bypass the section which selects contour levels automatically, as we have supplied our own levels.

(d) Enter the main loop. For each contour level do the following:-

  (i) Initialise variables to indicate that the current contour is of standard thickness and without annotation, and by-pass those sections which might alter these values if the labelling or thick-line options were being used.

  (ii) Initialise variables indicating the position of the first free space in the working arrays K, XY and K3 (see 3.4 for an explanation), and set the current contour height.

  (iii) For each element of the grid, check whether the current contour level lies between the bounds for that element stored in ZLIM. If not, there is nothing to do. Otherwise call subroutine SQUARE which, along with those routines which it calls directly or indirectly, calculates the conic sections within each triangle of the element, attempts to link them to others which have been calculated earlier, and plots any contours which have now been completed.

  (iv) If any contours in the data structure have not yet been plotted (this should only happen if the contour suppression feature has been used) call subroutine EMPTY which plots these contours.

(e) Bypass the crosshatching section of the routine and exit.

We now consider what happens on arrival at subroutine SQUARE. This subroutine deals with each of the four subelements in turn, beginning by calculating the bounds given in 2.6 for values taken by the function within the subelement. If the current contour level fails to lie between these bounds we proceed to the next subelement. Otherwise we pass on the eight values (four heights and four tangent intersection values) just calculated as arguments in a call to subroutine SUBSQ.

Subroutine SUBSQ does a simple calculation (as explained in 2.4) to evaluate the four tangent intersection values and one surface height needed within the subelement. Each triangle within the subelement is then checked against the bounds given by Lemma 2.1 and when it is possible that a section of the current contour may cross a triangle, subroutine TRICON is called to construct the conic section(s) across the triangle and add it to the data structure; however before TRICON is called it is necessary to calculate the intersections of the contour with the triangle's edges, a task which is carried out by calling subroutine EJCUT once for each of its edges. Care is taken to ensure that this subroutine is called no more than once to calculate the intersections along each edge within the subelement. (Some repeat calls may however be made during later calls to SUBSQ.)

Subroutine TRICON calls subroutine CONSEG which identifies the contour segments within the reference triangle and returns the endpoint-and-pole parametrisation for each separate piece of conic. Subroutine PLTCON is then called once for each section of conic.

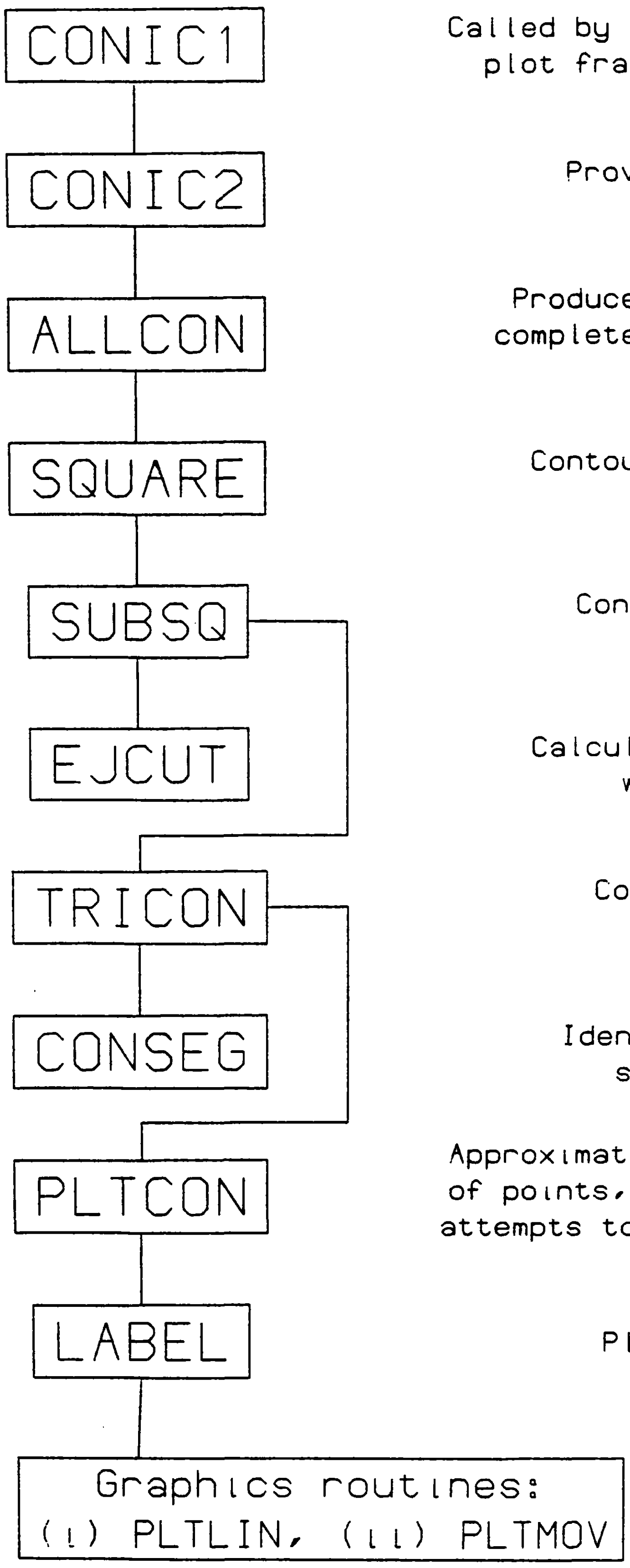
PLTCON is described in detail in Section 3.4; the first part of this routine calculates the straight line segments used to approximate the conic section, and the second part adds the conic section to the data structure and attempts to link it to partial contours already in the data structure. If the current conic section completes a contour, subroutine LABEL (see section 3.5) carries out the trivial task of plotting the contour, using the graphics routines which will have been supplied by the user's system.

To produce our simple plot we must therefore travel along a chain of several subroutines; this chain is illustrated by Figure 3.1.

## 3.3 Minor features of the package

Before examining major features of the CONICON package such as annotation and crosshatching we shall discuss some of those features which were simpler to incorporate but are nevertheless very useful.

FIGURE 3.1 HIERARCHY OF SUBROUTINES USED IN THE CONSTRUCTION OF A SIMPLE CONTOUR PLOT.



Called by user's program. Opens and closes plot frame and plots boundary of grid.

Provides a relatively simple interface to ALLCON.

Produces (with subroutines below) a complete contour plot with any number of levels.

Contours a single element at one level.

Contours a single subelement at one level

Calculates intersections of conic with one triangle edge.

Contours a single triangle at one level.

Identifies and parametrises conic sections within a triangle.

Approximates a conic section by a sequence of points, adds it to the data structure & attempts to link it to other conic sections in the data structure.

Plots a complete contour.

Supplied by user.
(i) Plots a straight line.
(ii) Moves plotter position invisibly.

### 3.3.1 Gradient Estimation

In practice the user is very often unable to calculate the true first order partial derivatives of his surface, and in such circumstances it is essential that a reliable gradient estimation routine should be available.

CONICON provides two such subroutines, called GRSET and GRSUB, and when gradient estimation is required one of these routines should be called prior to calling one of the master routines in the package. It is appropriate to call subroutine GRSET in normal circumstances and GRSUB in cases where the contour suppression feature is being used. Estimation is done in the most localised way possible, by fitting a parabola through a point and its two nearest neighbours in the relevant direction, and using the gradient of this parabola at the point of interest as our estimate. This technique preserves the contouring method's property of correct reproduction of quadratic surfaces, and given the piecewise quadratic nature of the approximant it seems the most natural method of gradient estimation. Subroutine GRSUB will not, however, use any point lying inside an area of local contour suppression for gradient estimation purposes. It sometimes therefore fits a straight line rather than a parabola to estimate the gradient at a point and in such cases the property of quadratic reproduction is lost in some areas of the plot.

Besides the usual case where the rectangular boundary of the map is probably a fairly arbitrary cut-off point, subroutine GRSET can cater for examples in which the surface can be thought of as a function over a cylinder or a torus and heights and gradients on opposite edges of the rectangle are therefore identical. In such cases the nearest neighbours of a point lying on the edge of the plot will still be situated in opposite directions.

The simplicity of this method of estimation belies its effectiveness. Figure 3.2 shows the superposition of two plots (both using a 31 x 21 grid

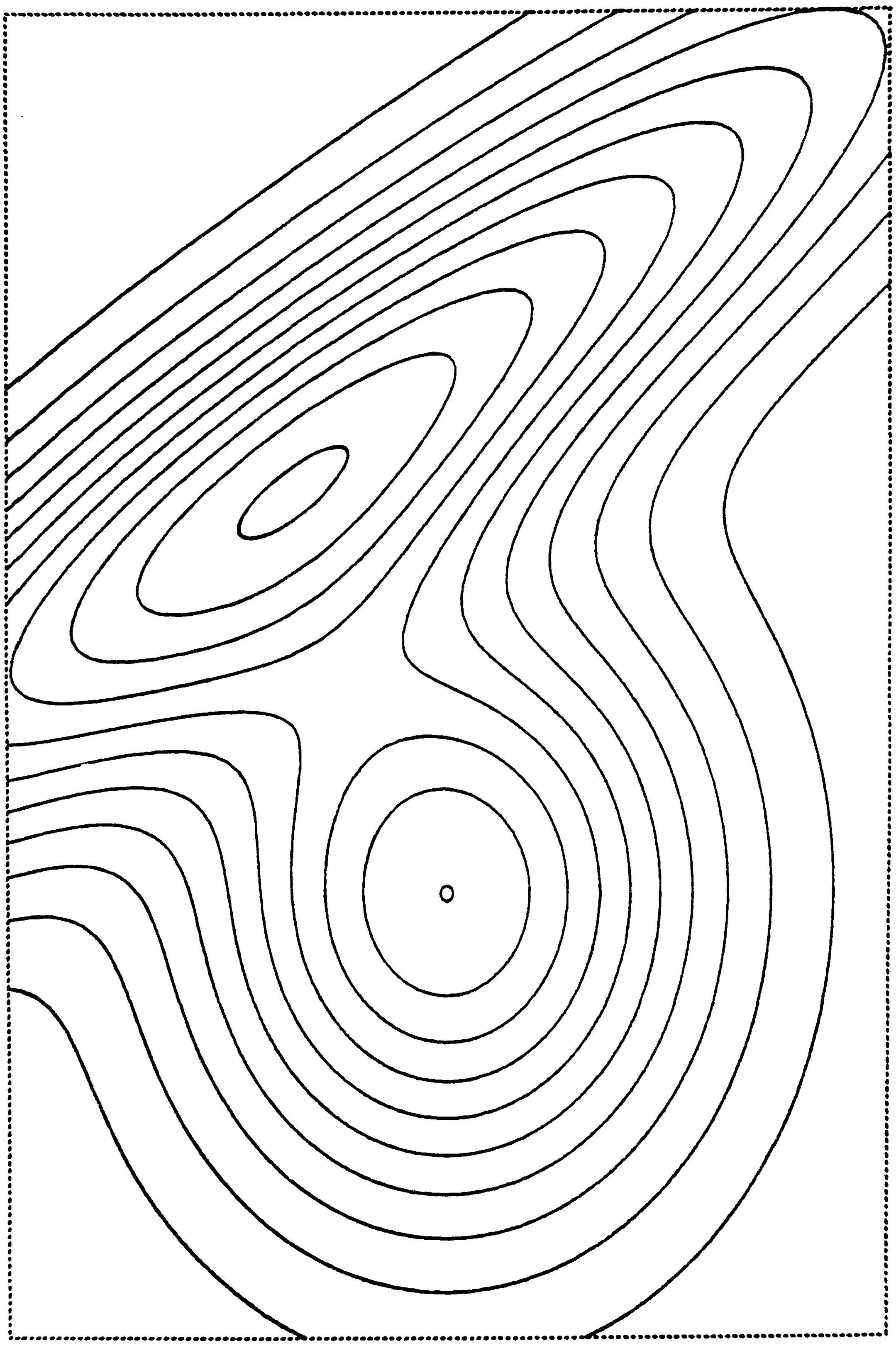


Figure 3.2 Standard example function, contoured using (a) true gradient values and (b) CONICON gradient estimates.

of points) of the same function (the function which was plotted in Figure 2.5), one using the true gradient values and the other using gradient estimates provided by subroutine GRSET. The only visible differences occur in some areas of the outermost contours where the line thickness appears greater than it should be.

## 3.3.2 Automatic selection of contour heights

The CONICON package offers the user the option of choosing contour heights himself, or of having them chosen automatically. Automatic choice of contour levels is carried out at an early stage of subroutine ALLCON, as soon as the bounds for values taken by the function within each element have been calculated. At the same time as these bounds are evaluated we calculate (in the obvious way) a pair of bounds for the complete region of interest. Contour levels (a number specified by the user) are then chosen at regular intervals to lie between this pair of values (alternatively the user may specify such a pair of values himself). The contour levels are chosen to be 'round' numbers, so far as this is possible by a subroutine, SCZZZ, originally written for scaling graphs by P.J. Green (pers. comm.).

Subroutine SCZZZ chooses contour levels to be integer multiples of any one of the following multiplied by an integer power of ten:-

1, 1.25, 1.5, 2, 2.5, 3, 4, 5, 6, 8

It therefore follows that it is not always possible to select exactly the number of contour levels requested by the user, but the routine will choose the greatest possible number of contour levels less than or equal to the number selected by the user.

## 3.3.3 Automatic choice of label length

In the CONICON package the user is free to choose how many decimal places the numbers which label his contours should carry; alternatively, he may leave the package to determine this for itself. One advantage of the latter course of action is that labels on different contours can then have different numbers of decimal places.

Subroutine WIDTH determines the number of characters (including decimal point and minus sign) in a label, whether or not the number of decimal places is chosen automatically.

The number of digits preceding the decimal place is determined easily, by examining the integer part of the logarithm (base 10) of the absolute value of the number. If the number of decimal places has been selected by the user then the only other information to be determined relates to the presence or absence of a minus sign and/or decimal point. Otherwise the routine must then determine the number of decimal places in the label. This is limited to a maximum of four. The following extract from subroutine WIDTH shows exactly how this number is determined (CT here represents the contour level itself, and NFRAC the number of decimal places).

```
  DATA TT, EPS/1.0E04, 5.0E-05/
  NFRAC = 0
  ACT = ABS(CT) + EPS
1 ACT = (AINT(ACT*TT))/TT - AINT(ACT)
  IF(ACT.LT.EPS) GOTO 2
  NFRAC = NFRAC + 1
  IF(NFRAC.EQ.4) GOTO 2
  ACT = ACT*10.0 + EPS
  GOTO 1
2 CONTINUE
```

## 3.4 Linking contour segments

It would of course be a simple task to ignore the continuity of the piecewise quadratic approximant and to plot each conic section produced by the package immediately after generating it. Indeed some of the more primitive contouring packages which employ piecewise linear methods do draw contours in such a manner, plotting individual straight line segments separately. However, a number of practical considerations dictate that such a policy should be avoided if possible; instead it is preferable to carry out a linking process of all conic sections at each contour level prior to plotting, and to plot complete contours without (in the case of a pen plotter) the pen having to leave the paper. The most important reasons for doing this are as follows:

(i) To save on 'pen-up' time on a graphics device - a significant consideration when a pen plotter is used.

(ii) To allow a sensible annotation algorithm to be incorporated into the package.

(iii) To allow the proper construction of broken-line contours.

The CONICON package therefore incorporates such a linking process as an obligatory part of the production of contour plots, and in this section we explain the data structure used to implement this feature and how it is updated by subroutine PLTCON each time a new conic section is produced.

### 3.4.1 Data structure for linking contours

It can be seen from the CONICON documentation in Appendix A that even the simplest routine for contour plotting, subroutine CONIC1, includes the working arrays XY, CONT, K and K3 in its argument list. These arrays are all connected with the data structure which is used in the process of linking conic sections into complete contours. Their functions are as follows:-

The array XY(2, NXY) holds the Cartesian coordinates of all the points used to approximate sections of the current contour across triangles, stored in the order in which they are generated. Thus when a new set of such points representing a conic section is generated early in subroutine PLTCON it is added to the next free space in XY. The first dimension of the array K(3, KTOP) provides a look-up table for the locations in XY of the end points of these conic sections; so if, for example, the first two conic sections in XY comprise 4 and 7 points respectively we will have K(1, 1) = 1; K(1, 2) = 4; K(1, 3) = 5 and K(1, 4) = 11.

The second dimension of K indicates the way in which conic sections are linked to each other. Entries are initialised to be zero and altered after the successful linking of pairs of conic sections. For example, if the 'bottom' end of conic section 2 is joined to the 'top' end of conic section 4, thenK(2, 3) = 8 and K(2, 8) = 3. In addition, if we find that the Nth segment endpoint lies on the boundary of the grid, we set K(2, N) = KTOP + 1.

The vector K3(NK3) can be dimensioned considerably shorter than the other three working arrays discussed here; the odd entries are used as pointers to those segment ends in the data structure which remain unlinked, while the entry in K3(2N) is the index number of the segment end at the opposite end of the chain of linked conic sections which begins at the Nth 'free' segment end. The segment endpoint indicated by K3(2N) will either lie on the boundary of the plot, or will also be available for linking. Information in the even entries of K3 is kept so that it can quickly be determined whether a complete contour has been linked up after the addition of a new conic section.

The third dimension of K simply performs the inverse transformation of odd entries in K3; thus if K3(2M-1) = N, then K(3, N) = 2M-1. This information is needed for successful updating of K3. The third dimension of K also performs a vital role in CONICON's garbage collection routine (see subsection 3.4.5).

Finally the array CONT(2, NXY) is used when a complete contour is linked and ready to be plotted. The coordinates of the points which form the contour are copied from XY to CONT in their correct order, and the contour is then plotted by subroutine LABEL. All contours are plotted as soon as they have been completed, beginning at the end nearer to the current plotter position. As they are copied from XY into CONT the values in the second dimension of K which have become redundant are flagged with minus signs. It then becomes possible to call an efficient garbage collecting routine which removes all the redundant information from the arrays XY and K; this permits major savings in storage space in examples where contours at all levels tend to be short and numerous.

We now present an example which illustrates the state of the data structure at a single instant during the construction of a contour plot. For simplicity we are contouring within a single element, which is illustrated by Figure 3.3. The dotted lines delineating the panels of the element are superimposed in order that the separate conic sections may be distinguished. Of course CONICON does not normally plot contours until they are fully linked, but here we plot all conic sections which have been generated regardless of their current state of linkage. The index numbers of all conic section ends are indicated in Figure 3.3; in this example we have interrupted the program immediately before conic sections in the NW subelement are calculated. At this stage eleven separate conic sections have been generated, two complete contours have been linked (and plotted) and there are two conic section ends which remain unlinked. KTOP in this example has been set to 200. The values in K3 and the second dimension of K at this stage are as follows:-

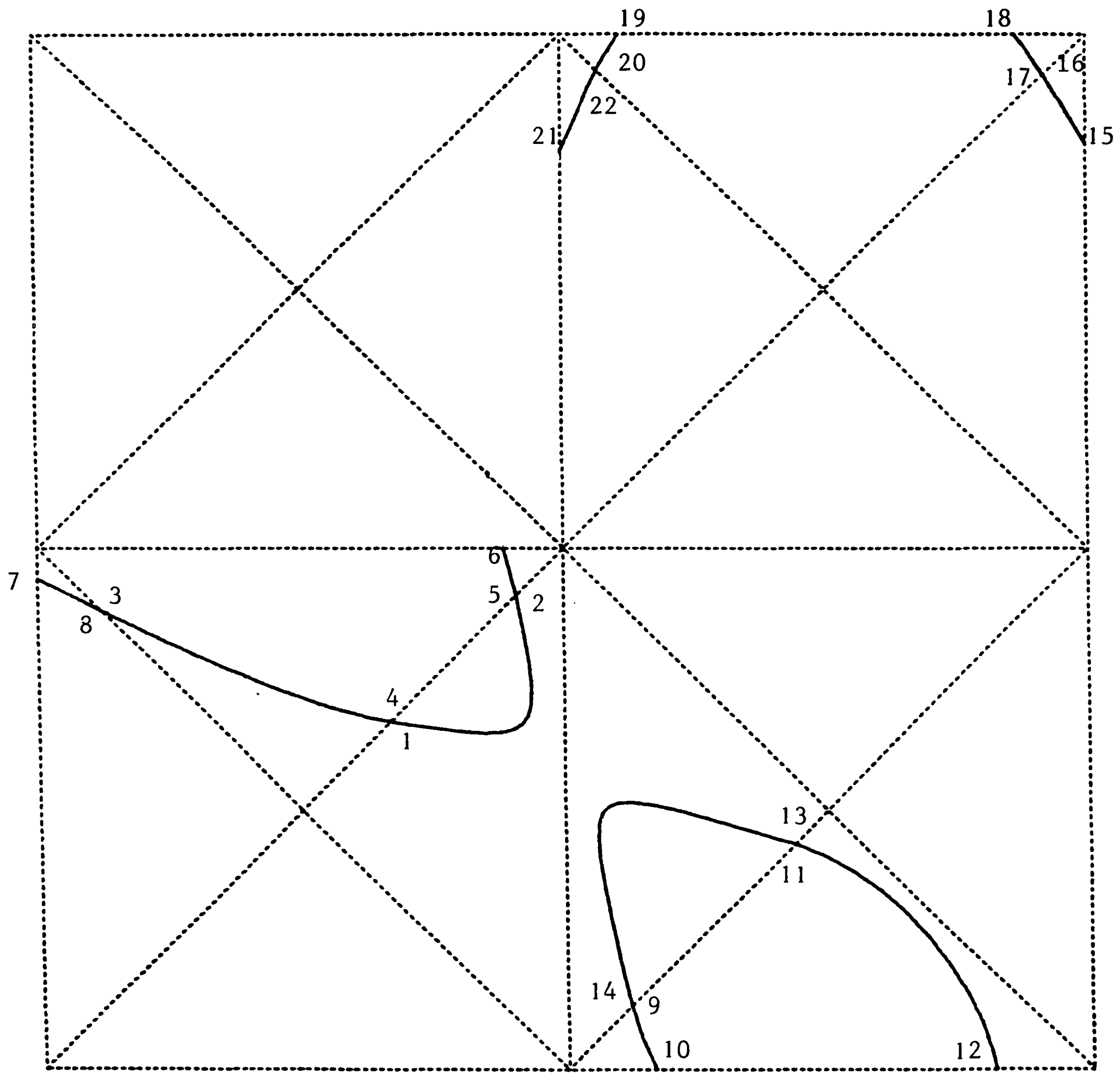


Figure 3.3 A single element example to illustrate the data structure used in linking contours.

|  |  |  |
|---|---|---|
| K(2,1) = 4 | K(2,9) = -14 | K(2,17) = -16 |
| K(2,2) = 5 | K(2,10) = -201 | K(2,18) = -201 |
| K(2,3) = 8 | K(2,11) = -13 | K(2,19) = 201 |
| K(2,4) = 1 | K(2,12) = -201 | K(2,20) = 22 |
| K(2,5) = 2 | K(2,13) = -11 | K(2,21) = 0 |
| K(2,6) = 0 | K(2,14) = -9 | K(2,22) = 20 |
| K(2,7) = 201 | K(2,15) = -201 |  |
| K(2,8) = 3 | K(2,16) = -17 |  |

K3(1) = 6

K3(2) = 7

K3(3) = 21

K3(4) = 19

### 3.4.2 How conic sections are linked

As was mentioned earlier, subroutine PLTCON carries out the tasks of updating the data structure and linking following the generation of a new conic section. The steps which PLTCON takes after it has approximated a conic section by a sequence of points are listed below (though for the present we shall omit reference to the rather complicated process of updating K3, which will be discussed separately).

1. Carry out a garbage collection if the new data will not otherwise fit into K and XY. If there is still insufficient space on return, STOP 620 or 621.
2. Determine whether either end of the conic section lies on the boundary of the plot, and store this information. If both ends lie on the boundary call LABEL immediately to plot the contour with or without annotation, and return.

3. Add all points of the conic section to XY, and make appropriate additions to the first two dimensions of K. At this stage the values added to K's second dimension will be 0 or KTOP + 1.
4. Return if no free segment ends are available for possible linking to the current segment.
5. If the top end of the new conic section lies on the boundary of the plot, go to (8). Otherwise search through all available segment ends and determine whether any should be linked to the top end of the new segment (allowing linking only if a pair of points is numerically identical). If so, make appropriate alterations to the second dimension of K; also use K3 to find the index number of the point at the far end of the newly extended chain of linked conic sections.
6. If we have succeeded in linking the top end of the new segment to another conic section, check whether a closed loop has been completed. If so, go to (10).
7. If the bottom end of the new segment lies on the boundary of the plot, then either (i) If the top end was linked to another segment, thereby completing a contour, go to (9)
   or (ii) return.
8. Carry out a similar matching process on the segment's bottom end to that attempted with its top end. If it cannot be linked to another segment, return. If it can be linked to another segment, check to see whether a contour has been completed. Return unless this is so.
9. Determine which end of the new contour is closer to the current plotter position (stored as (XPT, YPT) ).
10. Copy the various conic sections which form the new contour from XY into CONT in their correct order, with the first point being at the end nearer to (XPT, YPT). Flag values in the second dimension of K with negative signs as they become redundant. Reset (XPT, YPT) to

be the final pair of coordinates in the contour. Call LABEL to plot the contour and return.

### 3.4.3 Updating the array K3

The updating of K3 after a new conic section has been added to the data structure needs to be treated with some care, particularly if we succeed in linking one or both ends of the new segment to other conic sections.

When an endpoint of the new conic section has been linked to another conic section, K3 is immediately amended to take account of this. For example, suppose the new conic section is the 8th and we are successful in linking its bottom end to the top end of the 5th conic section in the data structure, which happens to be the third 'available' segment end in K3 i.e. K3(5) = 10. Then we will update K3 so that K3(5) = 16, indicating that the top end of segment 8 is now the third free segment end (whether or not it is indeed free). K3(6) does not require updating, because the segment end-point at the far end of the chain of conic sections is unaltered. However we must look to see if this segment endpoint is also available for linking:- if it is not (i.e. it lies on the boundary of the plot) then there are no more alterations to be made. Suppose though that it is free; we then look up its location in K3 using the value in the third dimension of K and find that this is, say, the first location. It follows that the value in K3(2) will be 10 ; this we must update to 16 to take account of the newly added segment.

We may however discover later that the top end of segment 8 had already been linked to another segment or alternatively that it lies on the boundary of the plot. In such cases we will therefore have to remove the third pair of entries from K3 (and in the former case a second pair of entries must also be removed from K3) and replace them by the final pair of non-zero entries in K3 before either returning or calling subroutine LABEL.

Thus, before leaving subroutine PLTCON we must carry out the following tasks:-

(i) Make one pair of deletions from K3 for each endpoint of the new segment which has successfully been linked to another segment.

(ii) Make one pair of additions to K3 for each endpoint of the new segment which we have failed to link to other segments (unless it lies on the boundary of the plot).

The net effect of this however, on the frequent occasions when a single end of the new segment is linked to another segment (and the other end does not lie on the boundary of the plot) is that no changes are made.

### 3.4.4 Retrieval of information from XY

CONICON's conic-following routines guarantee exact matching of conic section endpoints and for this reason the package always succeeds in correct construction of complete contours by the linking together of conic sections. However if the local contour suppression feature (see section 3.7) is used the package cannot always recognise when a contour has been completed: subroutine PLTCON can identify complete contours (a) which are closed loops or (b) whose endpoints lie on the (rectangular) boundary of the plot, but if the local contour suppression feature is used the package will usually generate some contours which fall into neither of these categories and will not be recognised as complete contours by the package. Therefore, in order that this does not result in contours being omitted from the plot, a check is carried out after all elements have been processed to determine whether NENDS (twice the number of 'free' segment ends in the data structure) equals zero. If NENDS is non-zero then not all complete contours have been recognised as such and it is therefore necessary to retrieve and plot the information still held in XY. This is done by calling subroutine EMPTY. This routine simply locates the segment ends which have not been linked, and copies each contour into CONT before calling subroutine LABEL, which plots it. When a contour has been plotted, the value in K3 corresponding to the end not yet located by EMPTY (if this does not lie on the boundary of the grid) is flagged with a minus sign in order to prevent contours from being plotted twice.

A second use for this subroutine, which is currently implemented only in the ECMWF version of CONICON, prevents the necessity of aborting a job if the dimensions of the arrays K or XY turn out not to be large enough: In the standard version of the package the user's program is forced to terminate at STOP 620 or STOP 621 if either of the variables KTOP or NXY turns out to be too small to allow all contours to be fully linked internally; however if subroutine EMPTY is called instead of aborting in such circumstances it will plot all partial contours generated up to that point and allow the job to be continued, with all space in XY and K once again free for use. This has little adverse effect on total CPU time: the only disadvantage is that annotation can become sparse if either KTOP or NXY is considerably below the ideal.

### 3.4.5 Garbage routine

As has been mentioned above, CONICON employs a garbage collecting routine to clear redundant information from the arrays XY and K when there is insufficient room for new information to be added to the free space in either one of these arrays. Subroutine GARB carries out this task in the following way:-

We begin by searching through the second dimension of K to find the location of the first piece of redundant information in XY (flagged as negative in K). We then find the next block of useful information in XY and transfer this down XY so that the first pair of values is moved to where the first pair of redundant values occurred, and so on. We continue locating blocks of useful information and shifting them down XY in this way until all the useful information on contour segments is located at the bottom end of XY, and IXY (the location of the first free space in XY) is then updated. The first dimension of K is also updated as this process is carried out; however K's second dimension remains unaltered at this stage. The third dimension of K is used to keep a temporary record of the movement of contour segments down the array XY. For example, if the seventh segment becomes the third segment after discarding four redundant segments, we will have K(3,13) = 5 and K(3,14) = 6.

We now move on to the updating of the second dimension of K. In cases where a segment end has not been linked to another segment end, values (either 0 or KTOP + 1) are simply transferred down the array. However, when a value in this array indicates that the conic section has been linked to another conic section, it is probable that the index of the segment end to which it is linked will have changed. The new index of that segment end is given by the value stored in the third dimension of K. After this process has been completed KBASE (the beginning of the next free space in K) is updated and we move on to the updating of K3.

K3 is updated in a similar way to the second dimension of K, using information on the changes in indices stored in the third dimension of K. Finally values are replaced in the third dimension of K using the information now in the odd entries in K3.

## 3.5 Plotting and annotation of contours

These functions are carried out by subroutine LABEL. Using a graphics interface conforming to the specifications outlined in the CONICON documentation, plotting of a contour without annotation is a trivial operation, whether thick or ordinary line styles are used.

Thus most of subroutine LABEL is devoted to finding the locations at which labels occur and calculating the correct points where plotting of a contour should end and begin in the locality of a label.

Subroutine ALLCON determines whether or not contours should be annotated and whether they should be plotted using thick lines, by examining first the value of II and then, if necessary, examining the values of ITH and ILAB once each at every contour level.

If it has been decided that the current contour requires annotation, the positions of the labels need to be determined. The algorithm which subroutine LABEL employs to select label positions is as follows:-

(i) Calculate a pair of critical values, C1 and C2 (C1 < C2), depending on the dimensions of the plot and the lengths (i.e. number of characters) of labels on the current contour.

(ii) Calculate the length of the current contour.

(iii) (a) If contour length < C1, plot the contour without labels.

(b) If C1 < contour length < C2, annotate the contour once only at its midpoint. (If the contour is a closed loop we regard the position where plotting begins and ends as its endpoints.)

(c) If C2 < contour length, provide the contour with two or more labels, the first occurring a short distance from where plotting starts and the remainder at equal intervals along the contour (the size of these intervals also depending upon the dimensions of the plot, etc.)

These choices of label position are all however subject to the constraint that a small rectangle surrounding each label should not overlap the boundary of the plot. If such an overlap would normally occur the current label and all subsequent labels are shifted in short steps along the contour until the rectangle around the current label falls completely within the plot, or the contour ends.

In the current implementation CONICON uses hardware characters for labelling and plots them all at the same (vertical) orientation, unlike a number of other contouring packages whose labels are plotted at orientations depending on the direction of the contours in the areas where they occur. It is felt that this policy makes CONICON's labels easier to read in general, except in cases where contours are closely spaced relative to label size and are near vertical. It is also felt that the algorithm outlined above for choosing label positions usually results in a highly satisfactory pattern of sites, and that this is borne out by the examples which appear in this thesis.

The steps carried out by subroutine LABEL are as follows:-

1. Move plotter position invisibly to the start of the contour. Jump to (8) if annotation is not required.
2. Calculate the length of the current contour, critical values C1 and C2 and hence the distance D2 of the next (first) label along the contour. If the contour is too short to be annotated proceed to (8).
3. Keep a running total of the lengths of the straight line segments which form the contour, until the total exceeds D2. Find the correct position of the centre of the label on the last straight-line segment.
4. If a small rectangle around the current label would overlap the boundary of the plot, make a small addition to D2 (assuming there will still be enough space on the contour for a label a little further along; if not, go on to (8)) and return to (3).
5. Keep stepping back along the contour from the label's centre, one straight line segment at a time, until a point is reached which

lies outside the rectangle around the current label. Then find the intersection of the rectangle edge with the straight line segment which crosses it. Plot the contour (with a thick or ordinary line style) from the current plotter position as far as this point, and then plot the label itself.

6. Now step forward along the contour until we find the straight line segment which leaves the rectangle around the current label and find its intersection with the rectangle edge. Move the plotter position here invisibly.
7. If the current label is the final one jump ahead to (8). Otherwise increment D2 and determine whether the next label will be the final one. Go back to (3).
8. Plot the remainder of the contour in a standard or thick line style.
9. End.

## 3.6 Crosshatching

A technique which has long been used in conjunction with hand drawn contour plots of relief, population density, rainfall etc. is that of crosshatching. This refinement is most commonly employed to achieve the effect of a progressive darkening of the map as the height of the surface increases and in such a form crosshatching is a useful aid to fast and easy assimilation of contour plots. The piecewise quadratic nature of the surface generated by our contouring method makes the automation of crosshatching a relatively straightforward problem to solve in our case, and this technique has therefore been incorporated into the CONICON package.

Broadly speaking, the problem of crosshatching can be subdivided into two distinct problems:- firstly the creation of the various patterns which might be required by the user, and secondly, finding all intersections of lines forming these patterns with the contours themselves. The former of these two problems has already been solved and implemented in the TILE 4 package (Sibson, 1980), which is able to crosshatch the area within an arbitrary convex polygon. The piecewise quadratic nature of the approximant means that the latter problem reduces to one of finding the intersections of straight lines and conic sections, which presents few difficulties. Of course the solution would be even simpler if a true piecewise linear contouring method were used; however to the author's knowledge no other contouring package suitable for vector graphics devices offers this extremely attractive and useful facility.

The CONICON package includes two fundamentally different crosshatching algorithms, both of which are described in some detail below. Both algorithms do however solve the two major problems described above in an identical manner, and we therefore precede discussion of the algorithms by a description of how the package tackles these problems.

### 3.6.1 Creation of hatching styles

In order to explain how the numerous crosshatching patterns available in CONICON are created, we refer to two subroutines, XHATCH and HATCH, from the TILE 4 package. With the obvious exception of the simplest style of hatching, patterns are created by superimposing two or more rasters of parallel, equidistant lines, lines being either solid or broken. Solid lines will usually be sufficient for the sort of progressive darkening of hatching styles mentioned above, but the ability to construct broken lines makes it possible to produce (at the cost of very little extra labour) quite elaborate patterns which may be useful alternatives in certain special applications. Subroutine XHATCH, the higher level routine, crosshatches the area within an arbitrary convex polygon by calling subroutine HATCH once for each raster of lines required. Indeed XHATCH is composed almost entirely of calls to HATCH, the particular calls which are selected on any one occasion depending on the code number of the crosshatching style chosen. The arguments of HATCH include variables (ANG, Q0, Q1, P0, P1, P2 and P3) representing the values of seven parameters which together specify the raster completely. The meanings of these parameters are explained in the comments at the beginning of the code for subroutine HATCH, and the appropriate part is reproduced here (with permission).

"ANG is the angle in radians (anticlockwise positive) between the (X, Y) axes and the (P, Q) axes. Hatch lines are produced parallel to the P axis. Q1 (assumed positive) is the spacing of hatch lines in the Q direction and Q0 is the offset of some line in the raster (visible or not within the polygon) from the origin. P3 is nonnegative. If it is zero, solid lines are produced, and P0, P1, P2 are ignored. If it is positive, it is taken as the gap length in broken lines, with P2, assumed positive if P3 is, as the dash length. On the line at Q, a dash starts at P0 + Q*P1 and extends in the positive direction. If any of the assumptions do not hold, no hatching is done".

Subroutine HATCH begins by finding Q values for all vertices of the polygon and is then able to determine the line of the raster lying within the polygon which has minimum Q value. Intersections of this line with the sides of the polygon are calculated and the line is then plotted, care being taken in the case of broken lines that dashes begin and end in the correct positions. Q is incremented in steps of Q1 and the remainder of the lines of the raster are plotted - in alternating directions to minimise 'pen up' time. Eventually Q exceeds the maximum value for any of the vertices and the hatching is completed.

### 3.6.2 Intersections with contours

The problem of finding the points where hatching lines and contours of the piecewise quadratic approximant intersect reduces to finding the intersections of a straight line and a conic section within a triangle. In CONICON, this is done by solving the equations of the conic and straight line in homogeneous coordinates. Since we have three coordinates and two equations the condition $x + y + z = 1$ is imposed to obtain a solution, thus specifying barycentric coordinates. As was stated in Section 2.6, the coefficients in the equation of the conic in homogeneous coordinates are simply the values at the vertices and the tangent intersection values on the triangle's sides. The equation of the straight line (in the form $lx + my + nz = 0$) is easily calculated from its intersections with the triangle's sides. We therefore derive a quadratic in one of the three variables x, y or z. If the discriminant is negative, the two curves do not meet; otherwise we test to discover whether each intersection lies within the triangle (i.e. all homogeneous coordinates are positive) and, if so, convert the homogeneous coordinates of the intersection back to Cartesian coordinates.

### 3.6.3 How algorithm A works

This algorithm ties the process of crosshatching closely to that of contouring, by employing a strategy which relies on information discovered in the contouring part of the process: namely, whether or not each triangle of the piecewise quadratic is traversed by the current contour.

As the contouring is carried out, lists of information (vertices, values at vertices and tangent intersection values) on those triangles which are crossed by the contour are built up in the arrays XD and ABC at virtually no extra cost (except in storage space). The algorithm maintains two such lists throughout its course, one list for each of the contour levels between which the next band of hatching is to be carried out. Thus contours must be plotted in order of height (by convention, ascending), and contouring and crosshatching at each level are carried out alternately. After contouring at any particular level has been completed, subroutine ALLCON calls subroutine AHATCH (c.f. XHATCH in TILE 4, which it resembles closely), one of its arguments being the code number for the current style of hatching. AHATCH first checks to see if both lists of triangles are empty: if so, either there is no hatching to do (so RETURN), or the complete area of interest requires hatching, a task best performed by subroutine XHATCH. Normally, however, some of the triangles of the piecewise quadratic will have been crossed by one or both of the contours currently being considered, and in such cases subroutine RASTA is called one or more times, once for each raster of lines required, with parameters set appropriately for the current hatching style. RASTA, the heart of algorithm A, is a long subroutine which produces a single raster of hatching across that part of the surface which lies between a pair of specified contour levels, or above or below a specified contour level.

The major stages of subroutine RASTA are outlined below. We describe here the case where hatching is carried out between a pair of contours: the differences which occur when hatching above or below a single contour level are obvious.

1. Calculate maximum and minimum Q values for all triangles in both lists. Rearrange the data in each list efficiently according to minimum Q values, using a Shellsort (Shell, 1959).

2. Enter the main loop. Find the Q value of the next (first) line of the raster. RETURN if this exceeds the maximum Q value for the plot. Otherwise calculate the intersections of the line with the boundary of the plot.
3. Find those triangles in the lists which are crossed by the current line (a simple task since they have been ordered according to their Q values).
4. For each triangle in each list which is crossed by the current line, find intersections (if any) with the current contours (see 3.6.2) and place these intersections in a list in the array XI.
5. Calculate the height of the surface at both ends of the current line. Determine whether these heights are consistent with the number (odd or even) of intersections with each contour. If we fail the test, shift the line a small, invisible distance 'upwards' by making a small addition to Q, then subtracting Q1 and going back to (2). However, omit this test if one of the current contour heights is very close to the surface height at an end of the current line. Normally, we pass the test and proceed to the next stage.
6. Order the list of intersections according to their P values.
7. If one of the two current contour heights is very close to the height of the surface at the end where we should begin plotting, reverse the direction of the line. Then determine whether the line should be begun with a visible or invisible movement of the pen from the edge of the grid.
8. Plot the line, normally in the opposite direction from that of its predecessor, in an "off-on" manner (switching on or off as each contour intersection is reached), using solid or broken lines. Then return to (2).
9. End.

The test carried out in stage (5) is necessary to protect against failing to detect (or double counting of) intersections which occur in close proximity to the boundary lines between individual quadratics. However the test is unreliable in cases where the height of the surface along its edge is very close to one of the current contour heights, so it is omitted in such cases.

It should also be noted that sections (4) to (7) inclusive may be bypassed if we find that the previous and current lines cross none of the triangles in our lists, and that none of these triangles are situated between the two lines.

### 3.6.4 How algorithm B works

In contrast with algorithm A, this algorithm completely divorces the two processes of crosshatching and contouring, and this allows for much greater flexibility in its use. The number of bands of crosshatching produced is entirely independent of the number and positions of contours in a plot, and it is also feasible using algorithm B to superimpose different bands of crosshatching. This makes it possible to achieve the effect of progressive darkening of hatching with many fewer calls to the equivalent of subroutine RASTA than would be necessary if algorithm A were used. It is even possible, if desired, to crosshatch a plot using algorithm B without carrying out any of the calculations required to plot the contours of the surface.

For each desired band of hatching subroutine CONXB2, the appropriate master routine, makes a call to subroutine BHATCH. Unlike in the case of AHATCH (algorithm A) it is not possible to determine immediately whether the whole grid should be hatched in this style (though using the bounds for heights of the surface calculated early in ALLCON we can sometimes conclude that no hatching is required). Like AHATCH, BHATCH decides on the number and nature of the calls which must be made to subroutine RASTB (c.f. RASTA) and carries these out. RASTB has much in common with RASTA, but there are some major differences. It can be divided into the following stages:

1. Initialisation of variables.
2. (Same as RASTA). Enter the main loop. Find the Q value of the next (first) line of the raster (RETURN if this is greater than the max. Q value for the plot) and calculate its points of intersection with the edges of the plot.
3. Follow the current line along from 'left' to 'right'. For each element of the grid which it crosses, check whether either contour height lies within the bounds for that element stored in ZLIM. If so, consider the four subelements in turn. If and only if (a) the subelement is traversed by the current line, and (b) the bounds for function values within the subelement contain either contour height, call subroutine SUBHAT which finds all intersections with the current contours within the subelement and makes appropriate additions to the list of intersections in XI. After each subelement has been considered (or if it is not necessary to consider the current element) proceed to the next element which the current line crosses and continue in this manner until the boundary of the plot is reached.

4. etc. Very similar to section (5) onwards of subroutine RASTA. Return to (2) when plotting of a line is completed.

5. End.

Subroutine SUBHAT finds those triangles within the current subelement which

(a) are crossed by the current line, and
(b) contain at least one of the current contour heights within their bounds,

and calls subroutine TRIHAT once for each such triangle. TRIHAT simply calculates intersections of a straight line and a conic section within a triangle (as explained in 3.6.2) and adds these to the list of intersections in XI.

### 3.6.5 Algorithm A vs Algorithm B

A discussion of the relative merits of the two crosshatching algorithms is included in the CONICON documentation (Appendix B) in the Section entitled 'Choice of crosshatching algorithm'. The previous two subsections help to explain how the differences arise: algorithm A requires a large amount of memory for grids of large numbers of cells because it stores information on all triangles which are crossed by either of the current contours in the arrays ABC and XD. Additional storage space is also required for maximum and minimum Q values for each triangle in each list (TQ) and for sorting (LV and TS). We might also expect algorithm A to be inefficient in terms of run time due to the necessity of sorting each list once for every raster of lines required; however other factors more than offset this. Most importantly, we can immediately 'home in' on those triangles which might contain intersections of contours and raster lines, discarding most of those which do not. In areas where raster lines are free of such intersections no time is wasted, and this is also true in cases where the whole grid requires hatching in a single style; so in this sense algorithm A is more foolproof than algorithm B.

The advantages of algorithm B over A do therefore not usually relate to faster run time, but to less extravagance in the use of memory and much greater flexibility. In some cases this increased flexibility can also lead to greater efficiency in the use of CPU time:- in particular in the common case where we wish to produce an effect of increasing darkness of the plot as the height of the surface increases, the opportunity to superimpose different levels of hatching can be taken to good effect. One minor problem related to such a strategy does however exist - the user cannot exercise quite the same degree of control over how gradual the changes in intensity are from one band to the next.

### 3.6.6 Crosshatching in combination with Annotation

A further refinement of the CONICON package is a facility which allows the user to incorporate both crosshatching and annotation within the same plot, in such a way that crosshatching lines respect the positions of labels by leaving a rectangular area around each label blank. This allows labels which would otherwise have been obliterated by crosshatching lines to be read with ease.

Such a facility must only be considered a minor embellishment to the package (indeed it might be argued that crosshatching and annotation are alternative means of achieving the same effect and are in no way complementary), but its implementation was a non-trivial task involving hundreds of lines of code (it allows annotation to be combined with hatching produced by either algorithm A or algorithm B) and we therefore give some details here. The additional steps required for this feature are as follows:-

(a) When plotting contours

For each label that is plotted, save (in the array ALAB) the coordinates of its centre and its length (number of characters).

(b) During subroutine RASTA or RASTB

(i) At an early stage find maximum and minimum Q values for rectangular boxes surrounding each label and store these (also in ALAB). Then reorder the list of rectangles efficiently according to minimum Q values.

For each line of the raster:

(ii) Determine which labels, if any, are crossed by the current line and place information on them in a separate array AN.

(iii) Find the Cartesian coordinates of the intersections of the current line and rectangles around labels, and replace those values in AN by these.

(iv) Order the pairs of intersections according to their minimum P values.

(v) If any overlaps exist between rectangles, remove these by reducing the number of intersections in the list.

(vi) Combine the lists of contour intersections and label intersections (discarding some) into a final list of 'effective intersections' in the array XI. Plot as before.

Crosshatching accounts for a large proportion of the code in CONICON: at least half the code is devoted solely to crosshatching. There are three major contributing factors to this:- the use of two alternative crosshatching algorithms; the wide range of hatching styles available; and the capability to handle crosshatching and annotation simultaneously.

## 3.6.7 Examples

We conclude this section by presenting examples of CONICON plots which use the crosshatching facility. As explained in Appendix A, a total of 51 styles of crosshatching are currently available, of which the final six styles were added to those already available in TILE 4 by the author.

Figure 3.4 shows a single seamed element contoured and crosshatched using some of the more sophisticated styles available. It is also an example of the use of crosshatching and annotation in combination, and utilises the feature of automatic choice of label lengths to allow differing numbers of decimal places in labels on contours at different heights. The crosshatching in this plot was produced using algorithm A, which carries out the task more cheaply than algorithm B and does not require significantly more storage space in single element examples.

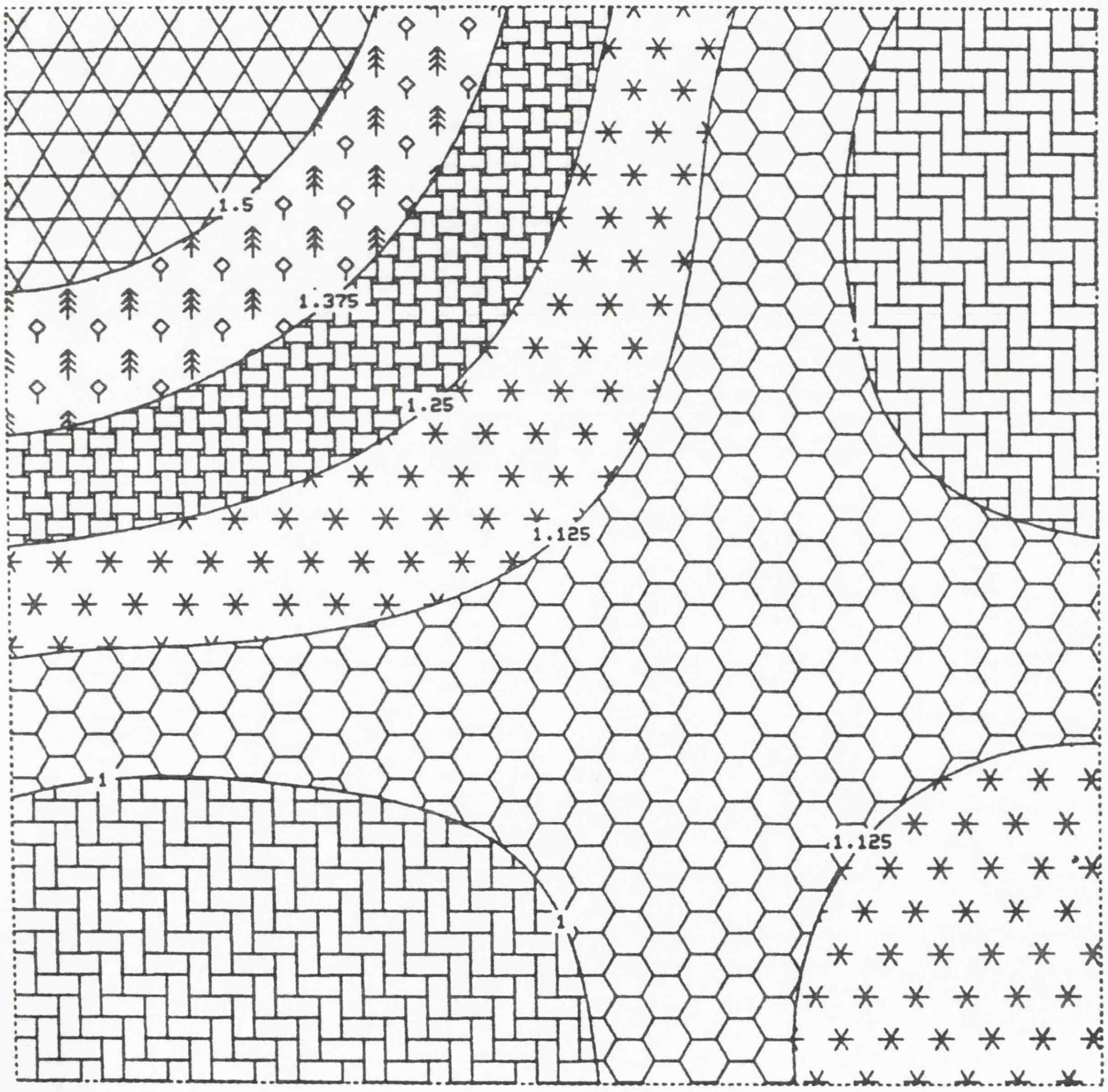


Figure 3.4 A single element showing some of the available hatching styles; contour annotation also included.

Figure 3.5 shows a contour plot in which the effect of progressive darkening of hatching styles has been achieved by the superposition of hatches using algorithm B (a similar plot produced by algorithm A turned out to be significantly more expensive). The function being plotted is one of several utility functions produced from data provided by A. Francescon from the Dept. of Operational Research at the University of Sussex. The original data were value-only and lay on a square grid with missing values. Therefore an interpolant was constructed, using the $C^1$ Natural Neighbour method (Sibson, 1982) to fill in missing values as well as gradients and this was contoured over the original grid, that is a grid of 36 x 24 points or 35 x 23 elements.

## 3.7 <u>Local suppression of contour plotting</u>

In many applications the user wishes to produce a contour plot of his function over a rectangular area; but sometimes a rectangular plot may be wasteful, or inconvenient, or even meaningless. For example, if the variable being investigated is the concentration of a particular organism within a circular pond, extrapolation to areas outside the pond will be totally inappropriate. In such cases we would prefer to define an M x N grid of elements as before, but only to contour those cells of the grid which lie completely within the area of interest. In addition we might wish to suppress contouring within some internal areas of the plot if, for example, one or more islands were present in the pond.

CONICON has a facility which allows the user to suppress contouring within a subset of grid cells of his choice. Implementation of this feature within the package was a straightforward task: the array ZLIM (2, MM, N) which stores bounds for values attained by the surface within each cell of the grid is used to double up as an indicator variable specifying those cells within which contouring should be suppressed. Thus

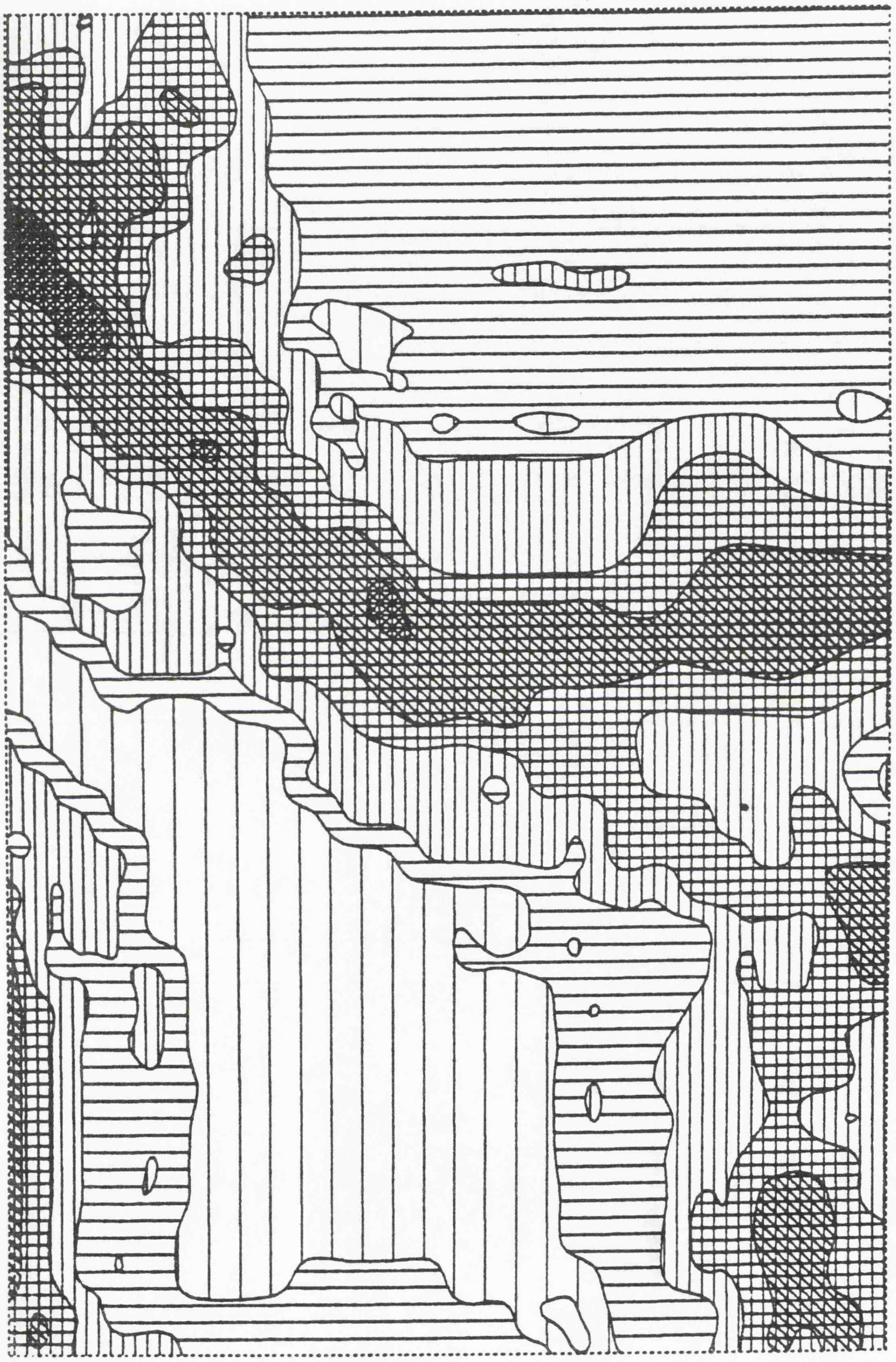


Figure 3.5 A complete contour map illustrating progressive darkening of hatching styles (algorithm B).

if the user indicates that he wishes to use the local contour suppression feature he is required to set values in the array ZLIM in the following way:- if he wishes the (I, J)th cell to be contoured he should set ZLIM(1, I, J) less than or equal to ZLIM(2, I, J); if not, ZLIM(1, I, J) should be set greater than ZLIM(2, I, J). When the stage of calculating bounds for each cell is reached, bounds are only calculated and made to replace the existing values in ZLIM in cases of the latter type; therefore when we come to the stage of contouring the (I, J)th element we simply check that the contour level C and the values in ZLIM satisfy ZLIM(1, I, J) < C < ZLIM(2, I, J) and only in such cases do we proceed with a call to subroutine SQUARE.

In a significant number of examples users may wish to restrict contouring to the area within a convex polygon; CONICON incorporates a subroutine called CONVEX which will automatically set suitable values in the array ZLIM to suppress contouring outside a convex polygon specified by the user. The polygon is defined by a number of linear constraints of the form $ax + by + c \leq 0$; all values in ZLIM are initialised as 0.0 by the routine and if the (I, J)th cell fails to satisfy one or more of the constraints ZLIM (1, I, J) is re-set to 1.0.

CONICON also contains a subroutine called BORDER which plots the boundary of the part of the grid lying inside the same convex polygon within which contouring has taken place. The subroutine relies on the convexity of the polygon and can be broken down into four stages:- in the first it zigzags up and from left to right; in the second stage up and from right to left; then down and from right to left; and finally down and from left to right.

As has been mentioned above, subroutine EMPTY will often be called upon to plot a number of contours in plots of this type and the arrays K3, K and XY should therefore be dimensioned longer in such examples. It should also be noted that labels cannot always be prevented from overlapping the boundary of the plot.

The crosshatching feature has not yet been combined with local contour suppression, but conceptually the problem appears only a little more difficult to solve than the crosshatching problem for a rectangular area. An algorithm similar to algorithm B would seem appropriate, since it would be necessary to trace each line of the raster from cell to cell, checking whether the area within each new cell has been contoured or deliberately left blank. The algorithm would have to evaluate the height of the surface at every point where a line of the raster crossed the boundary between a blank cell and a contoured cell. Every such point could then either be treated as an additional intersection or discarded, depending upon whether or not the height of the surface at the point lay within the band of hatching.

Figure 3.6 shows a plot in which subroutine CONVEX has been used. The function is the $C^1$ Natural Neighbour Interpolant constructed from another data set provided by A. Francescon. Subroutine BORDER has been used to plot the boundary of the contoured area and the polygon itself is also indicated.

It should be noted that, useful though the contour suppression facility is, it is still nothing more than a selector facility for plotting or not plotting in each grid cell. Contouring exactly over an arbitrary (in practice polygonal) region, even in the convex case, is not yet supported by the package; equally, there is nothing to prevent the future development of such a facility.

## 3.8 Gradient contouring and marking of stationary points

One minor criticism which may be levelled against contour plots produced by CONICON is that the high visual quality of these maps may delude the user into believing that they must always give an accurate representation of the surface being considered. Indeed because we use

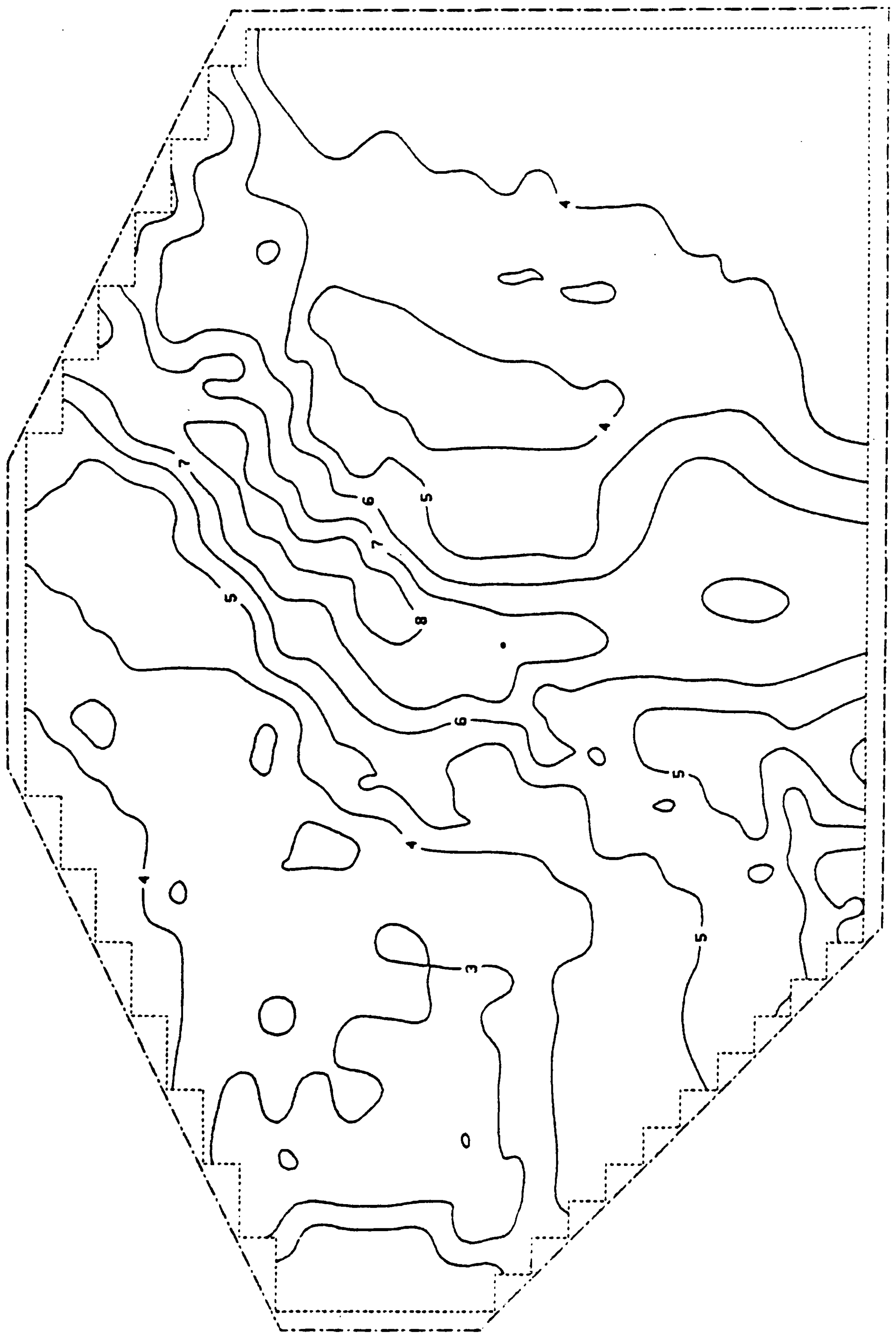


Figure 3.6 An example to illustrate local contour suppression.

gradient as well as value data CONICON plots are likely by and large to be accurate over most of their area; but in locations where the gradients of the surface are small and the Implicit Function Theorem (see Section 1.1) is close to breaking down, contours produced by this or any other package should be treated with a little more caution.

In Chapter 5 we discuss just how serious this problem really is, and conclude that there are grounds for believing that it is not very severe. Nevertheless some methods of reducing the errors in the contours in such areas are presented in Chapters 5 and 6. These methods have not however been included in the CONICON package, but the package does offer the facility to produce a contour plot of the squared magnitude of the gradient of the piecewise quadratic approximant. Gradient plots are produced by calling either of the master routines CONGR1 and CONGR2 contained in CONICON; a study of such a plot will give the user some indication of those areas in which contours of the function itself may be unreliable.

Plotting of the gradient of the approximant function is possible within the framework of CONICON because the approximant function is piecewise quadratic and continuously differentiable: the former property means that the partial derivatives of the surface vary linearly over triangles, and across such areas contours of the gradient will therefore be conic sections. The latter property means that contours of the gradient, though not continuously differentiable, will be continuous and can therefore be linked by subroutine PLTCON.

CONICON therefore produces plots of gradients in very much the same way as it produces ordinary surface plots; from TRICON downwards the subroutines employed in their production are identical. One difference which occurs is in bounds for values taken by the surface within elements, subelements and triangles. Since the first order partial derivatives over a triangle vary linearly, they can be represented by an expression of the form

$$\nabla f = \underline{b} + C\,\underline{x} \tag{3.1}$$

So $$\nabla f^T \nabla f = \underline{b}^T\underline{b} + 2\underline{b}^T C\,\underline{x} + \underline{x}^T C^T C\,\underline{x} \tag{3.2}$$

Now $C^TC$ is positive definite, so a local minimum of the gradient may occur anywhere within a triangle, but the maximum value within any triangle must occur at one of its vertices.

Thus, in order to find the maximum value taken by the gradient within either an element or a subelement we need only look at the gradients at the vertices of the triangles which form that construction. Bounds for the minimum value, whether exact or not, are much more difficult to calculate: we can of course still use the bounds for triangles given by Lemma 2.1, but the logic used in constructing bounds for larger areas no longer holds. Thus we have to consider all height and tangent intersection values within an element or subelement in order to obtain a non-minimal lower bound for the gradient within that construction. (When contouring the gradient of a function, the tangent intersection value at the mid-point of a line in terms of the partial derivatives $(s_L, t_L)$ and $(s_R, t_R)$ at either end is simply $s_Ls_R + t_Lt_R$.)

We now present an example of the use of this technique. Figure 3.7 shows a plot of the gradient of the function contoured in Figure 2.5, etc. As we might have expected, the areas where the gradient is smallest occur around the function's two peaks, the saddle point and near the perimeter of the plot. Since we have plotted few contours round the outside of the plot in Figure 2.5, the low gradient in this area is unlikely to worry us. It is indeed a common feature of gradient plots that they tend to indicate areas where few contours have been plotted as least reliable; this is a result of the very nature of gradient plots and the fact that in most maps contour levels are chosen at regular intervals.

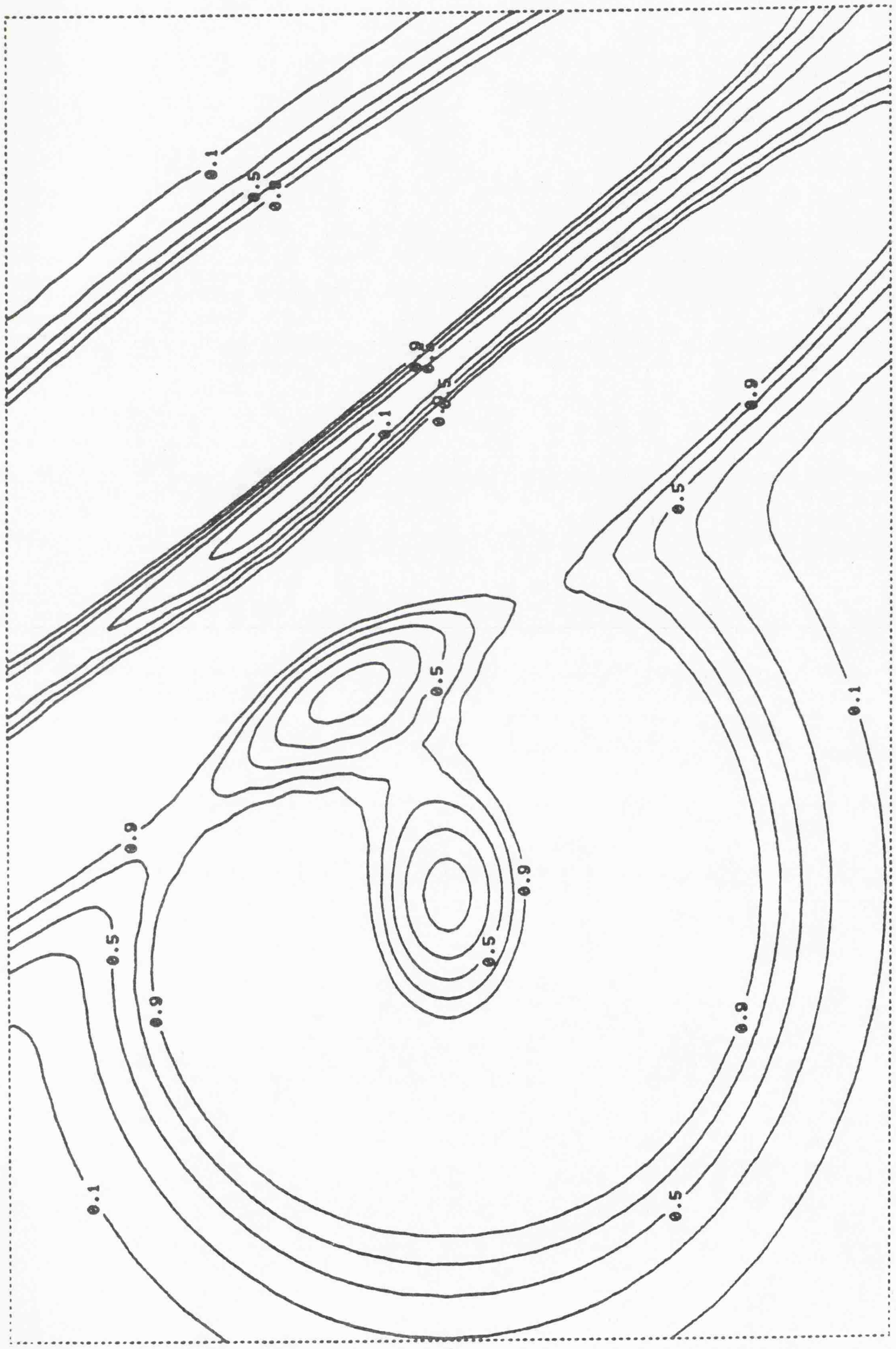


Figure 3.7 Gradient contour plot of standard example function.

Although we have not implemented this, it would in principle be fairly straightforward to use the gradient function to control contour plotting of the original function. For example, contours in areas of low gradient could be omitted or plotted as dashed lines to emphasise uncertainty. Also, in areas of very high gradient, subsidiary contours (e.g. those between "thick line" levels) could be omitted to avoid overcrowding. Such techniques are common in manual cartography.

Another use of the gradient function might be to improve annotation, to prevent labelling occurring in areas of high gradient where contours are close together and labels might become obscured.

The last of the facilities in CONICON which we describe, and one which is closely related to plotting the gradient of the approximant function, is the calculation and plotting of its stationary points. This is useful in certain applications, particularly in meteorology for isobar plots, where the labelling of 'H's and 'L's at highs (local maxima) and lows (local minima) respectively is common practice.

Use of the piecewise quadratic approximant makes the calculation of stationary points a relatively simple affair. No more than one stationary point may occur within a single triangle of the approximation and the simple form of the approximant within an individual triangle means that there is little difficulty in discovering the locations of such points. It is possible to calculate positions of stationary points directly as a fairly simple function of the six parametrising values on the triangle's edges (see for example Marlow and Powell (1976), p.10) but in examples where these values are large and close together a considerable amount of numerical instability is introduced into the calculations. CONICON therefore employs an alternative procedure, which makes use of the property that all the triangular panels of the approximant are right angled (and isoscles) and therefore the homogeneous coordinates corresponding to the two $45^{\circ}$ vertices (x and y by convention) at any point are simple linear transformations of the Cartesian coordinates of that point. We therefore regard these coordinates as Cartesian coordinates during our calculations for convenience. The steps required to determine the position and type of a stationary point within a triangle are then as follows:-

1. From the six parametrising values calculate partial derivatives in "x" and "y" directions at each vertex.
2. Determine whether both partial derivatives take at least one positive and at least one negative value at the triangle's vertices. If not, return.
3. Calculate the pairs of points along the triangle's sides where x and y derivatives respectively are zero.
4. If the four points just determined are not situated in such a way that the straight lines $\frac{\partial f}{\partial x} = 0$ and $\frac{\partial f}{\partial y} = 0$ meet within the triangle, return.
5. Calculate the point of intersection of the lines $\frac{\partial f}{\partial x} = 0$ and $\frac{\partial f}{\partial y} = 0$, and convert to true Cartesian coordinates.
6. Calculate the three second order partial derivatives of the quadratic and use these to determine whether the stationary point is a minimum, a saddle point or a maximum, and return.
7. End.

An example of the use of this facility appears in Chapter 4.

# CHAPTER 4

# MISCELLANEOUS APPLICATIONS

## 4.1 Introduction

In this Chapter we illustrate the wide variety of applications of the piecewise quadratic contouring method by applying it, as implemented in the CONICON package, to a number of data sets arising in disciplines as diverse as statistics, metallurgy, meteorology and geology.

It would of course be beyond the scope of this thesis to attempt to explain the significance of most of the surfaces which are contoured in this Chapter, and therefore we do not in general provide more than the minimum amount of background information about the data sets - most of the examples are presented to illustrate the high quality of the quadratic contouring method, the facilities offered by the CONICON package and its capacity to cope with the largest and most complicated data sets; some examples also afford a means of comparing alternative contouring packages with CONICON directly.

We begin in Section 4.2 by studying further the bivariate probability density estimate which was contoured, not entirely successfully, in Figure 2.11. This density estimate arose from a metallurgical application and the method of its construction and the significance of its appearance are amply explained in Silverman (1982) - our concern is merely with improving the accuracy of the plot.

Section 4.3 shows an application in statistical methodology:- we investigate a series of two-parameter empirical likelihood functions arising from simulated data, using a model suggested by Professor M.

Aitkin (pers. comm.). In these examples we do attempt to provide some explanation of the behaviour of the functions which we contour, and we find that contouring is a particularly useful means of improving our understanding of them - probably much more useful than any alternative method of display.

In Section 4.4 the CONICON package is used to contour data sets arising in meteorology, provided by the European Centre for Medium-range Weather Forecasts. Although these data sets comprise grids of up to 240 x 61 data sites the package is shown to cope with them quite adequately. These data sets also provide an excellent means of illustrating many of CONICON's features; and we include some examples which highlight the further improvements incorporated in the ECMWF version of the package.

Finally we use CONICON to contour a small assortment of published data sets in order to compare its plots directly with plots produced by other contouring packages. In all cases CONICON appears to produce plots of a quality at least as high as that attained by its rivals.

## 4.2 Contouring bivariate density estimates

We now return to the bivariate probability density estimate constructed by Dr. B.W. Silverman, a plot of which appeared as Figure 2.11. A full explanation of the variables represented by the x and y axes and the manner of construction of the estimate (in this case window width = 2.0) is presented in Silverman (1982); here we are concerned simply with the accuracy of the contour plot as a representation of the underlying surface.

It should be noted however that this density estimate was constructed from nearly 15,000 observations and therefore each evaluation of the height and gradient of the surface at a point requires a considerable amount of computation (though fortunately gradients involve little extra cost after calculation of heights); it is therefore important that as coarse a grid as possible should be used as input data for CONICON. For this reason, and because the error involved in using the seamed quadratic element is known to be of order $h^3$ (see Chapter 5 for further details), Silverman recommends the use of the seamed quadratic element not only for contour construction but also as a means of interpolation from true values and gradients on a coarse grid to estimated values on a finer grid which will allow density estimates to be displayed by the alternative means of perspective block diagrams.

We have already seen that the very coarse grid chosen by Silverman, one of just 25 elements, results in some very unnatural-looking behaviour on the outer (lowest) contour on the plot. We speculated in Chapter 2 that this behaviour was the result of near-breakdown of the Implicit Function Theorem (see Section 1.1) around the edge of the plot: it should be noted that gradient values on the boundary of the grid are all either zero or very close to zero; the contours begin at the 0.001 level and are plotted at intervals of 0.002 - thus the highest contour level is 0.027.

The most obvious way to improve Figure 2.11 is of course to use a finer grid, but as we have seen this is costly in terms of generating the data (the additional time involved in contouring itself is of lesser importance), so if an alternative method which did not involve use of a finer grid were to prove satisfactory, this would be preferable. Probably the only alternative method which does not rely on a different grid is to take a strictly monotonic transformation of the data and to plot transformed contour levels; such a process will of course have no effect on the true contours of the surface, but can have an adverse or beneficial effect on contours produced by the piecewise quadratic method, depending upon the suitability of the transformation. We will need to use a transformation which allows us to retain correct gradients at the grid points, or it is very unlikely that any improvement will be made. The obvious candidate would appear to be the logarithmic transform, as this will clearly remove the flatness near zero which appears to be the main source of trouble. However a log transform cannot be carried out on the data as it stands, because some data values are zero; therefore we add a small constant $\varepsilon$ (= 1.0e-06) to each z value before taking the log transformation, and to obtain gradient values we simply divide the partial derivatives of the true surface by $(z + \varepsilon)$ at each point.

The contour plot which results from this process is illustrated by Figure 4.1. This shows apparent success in that most of the unwanted oscillations have been eliminated from the outermost contour. However at the same time we have clearly altered the appearance of the map around the density estimate's mode: in this area contours appear less rounded than the corresponding contours in Figure 2.11, having apparently been stretched to some extent along a line running from the bottom left to top right hand corners of the plot, so that contours in the plot as a whole are more uniform in shape. It is impossible to tell whether or not this is an improvement over Figure 2.11 unless we know the appearance of the true contours of the surface; thus we have decreased grid size by a factor of two and plotted the contours of the (untransformed) surface over an 11 x 11 grid of points in Figure 4.2. This plot clearly has much more in

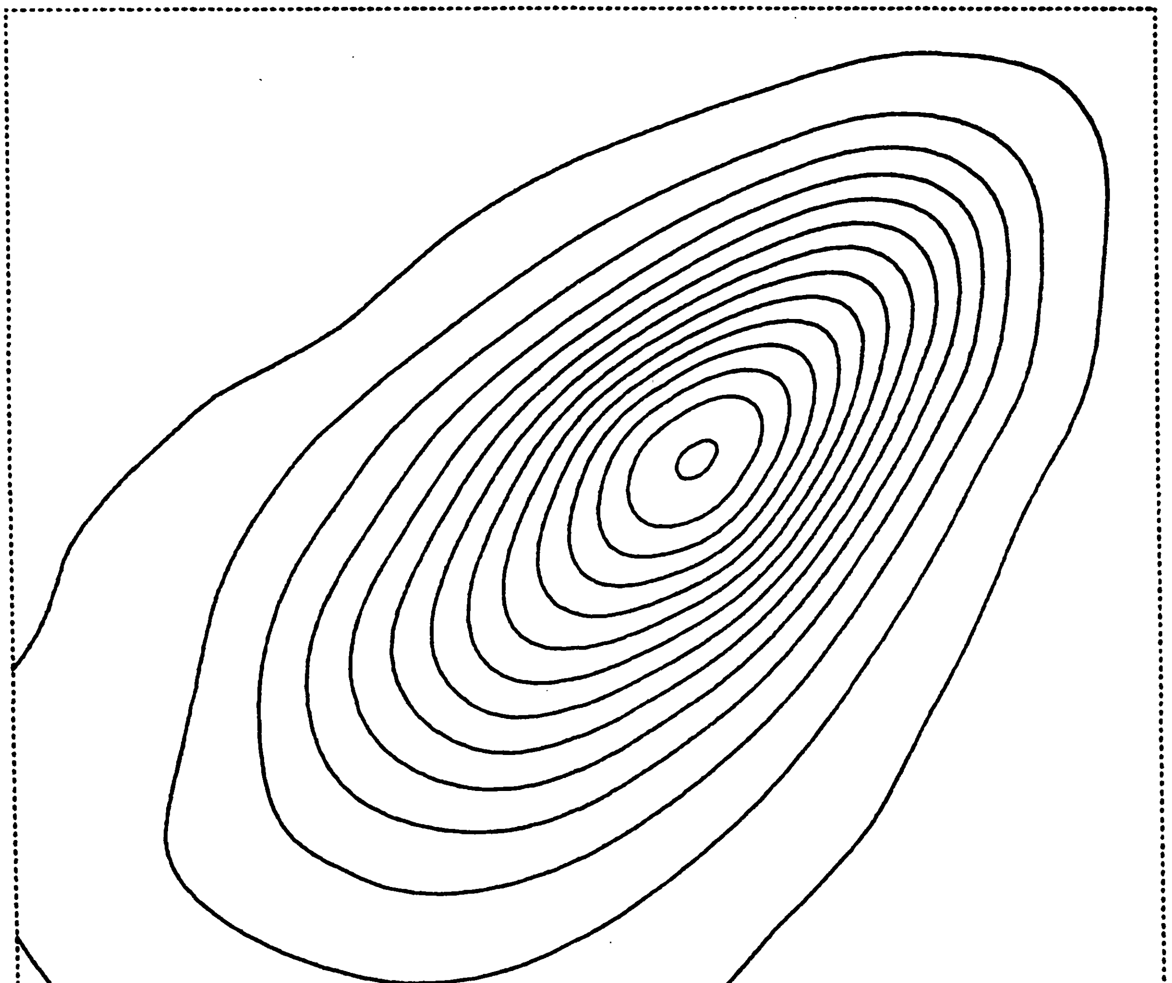


Figure 4.1 Logarithmic transform of a bivariate probability density estimate (5 x 5 grid of elements).

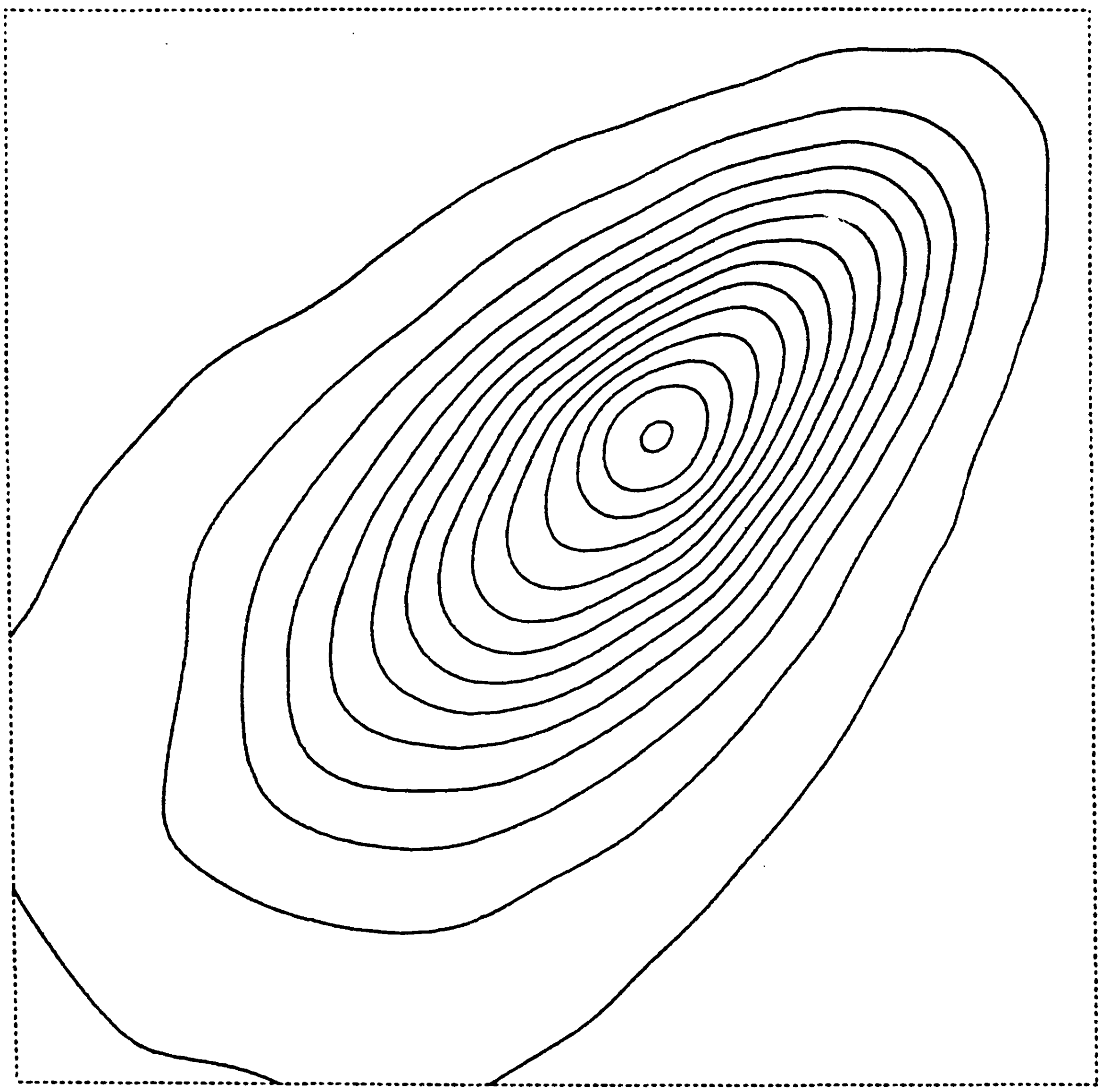


Figure 4.2 Untransformed probability density estimate (10 x 10 grid of elements).

common with Figure 4.1 than with Figure 2.11 and we can therefore conclude that use of the log transform has led to a considerable improvement; indeed it appears that if this particular example is a typical one (and plots of other window widths offer no evidence to the contrary) then the extremely cheap method of using a 6 x 6 grid of points and a log transform is likely to be suitable for contouring most bivariate probability density estimates, provided the window width (which controls the smoothness of such estimates) is not excessively small.

It should be borne in mind that in an example such as this the contours produced by a piecewise linear contouring method over a grid of such coarseness would be so poor as to be unacceptable even to someone accustomed to using such contouring methods; thus the production of even barely acceptable contours using such a method would still be a much more costly process than the production of highly acceptable contours with the method recommended above.

## 4.3 Looking at likelihood functions

In this section we use CONICON to look at surfaces which arise in a wholly statistical application, a two-parameter likelihood model formulated by Professor M. Aitkin (pers. comm.). The model which we are investigating is a Normal mixture model: data are assumed to arise from a pair of Normal distributions, one of which is the Standard Normal (i.e. mean = 0, standard deviation = 1) and the other, the contaminating distribution, is Normal with mean = $\mu$ and standard deviation = 1. Our two unknown parameters are the distance parameter $\mu$ and the mixing proportion, denoted by $\lambda$.

The likelihood function $L(\mu, \lambda|y)$ for a single observation $y$ is thus proportional to

$$\lambda \exp [ - \tfrac{1}{2} (y - \mu)^2] \quad + \quad (1 - \lambda) \exp [- \tfrac{1}{2} y^2] \qquad (4.1)$$

However it is more convenient to consider log likelihood (as this function is much easier to analyse mathematically); therefore for a vector $\underline{y}$ of n observations we have log likelihood:

$$\log L (\mu, \lambda|\underline{y}) = \sum_{i=1}^{n} \log \{\lambda \exp [- \tfrac{1}{2} (y_i - \mu)^2] + (1 - \lambda) \exp (- \tfrac{1}{2} y_i^2)\} \qquad (4.2)$$

plus a constant term.

The first order partial derivatives are then

$$\frac{\partial \log L}{\partial \lambda} = \sum_{i=1}^{n} \left[ \frac{1 - \exp \{\mu(\tfrac{1}{2}\mu - y_i)\}}{\lambda + (1 -\lambda) \exp \{\mu(\tfrac{1}{2}\mu - y_i)\}} \right]$$

$$\frac{\partial \log L}{\partial \mu} = \sum_{i=1}^{n} \left[ \frac{y_i - \mu}{1 + \frac{(1-\lambda)}{\lambda} \exp \{\mu (\tfrac{1}{2}\mu - y_i)\}} \right] \qquad (4.3)$$

Given any data set $\underline{y}$ which is assumed to satisfy our model, the (log) likelihood function tells us which combinations of the parameters $\lambda$ and $\mu$ are most likely to have caused such a data set to arise. In general we are not particularly interested in the magnitude of the likelihood at any point; more important are the shape of the surface and the positions of its local maxima.

The data used to generate the likelihood functions plotted in this section did not arise from a 'real' source, but were simulations from the NAG pseudo-random number generator (Numerical Algorithms Group, 1982), which uses the algorithm of Brent (1974). By providing data sets with known values of $\mu$ and $\lambda$ we hope to assess how the likelihood functions ought ideally to appear. This should aid our understanding of likelihood functions arising from data with unknown parameter values and might also help us to identify data sets not satisfying our assumptions.

In all likelihood plots presented in this section the horizontal axis is used to represent $\lambda$ varying from 0 to 1, and the vertical axis to represent $\mu$ as it varies between -2 and 3.

However before we begin to examine individual plots a minor but unfortunate deficiency of the quadratic contouring method which was brought to light by the likelihood functions in this section must be pointed out:- when investigating this model initially it was discovered that in nearly all the contour plots of likelihood functions produced by CONICON a peak covering only a small area but higher than the maximum value attained by the surface elsewhere appeared within the element in the top right hand corner of the plot, and in one case spread to a neighbouring element; this phenomenon occurred in an area throughout which the surfaces should all have been rapidly decreasing as both $\lambda$ and $\mu$ increased. Examination of the gridded data in this locality revealed no errors, and it eventually became apparent that the cause of these anomalies was behaviour similar in some characteristics to the familiar 'Gibbs phenomenon' in Fourier analysis: the piecewise quadratic method of approximation was incapable of handling in a sensible way the very large (negative) gradient in the $\lambda$ direction in the extreme north eastern corner of the plot - the only way that this could be accommodated was by inserting a spurious peak in that area of the surface.

To explain this further we consider the problem in a single dimension only, looking at the line segment which forms the 'top' edge of the element in this corner of the plot. Suppose this element is of length h with values and gradients (zl, sl) and (zr, sr) at its ends; then the value at the midpoint of the segment can easily be shown to be $\frac{1}{2}(zl + zr) + h/8(sl - sr)$. It can therefore be seen that, whatever the (fixed)values of zl, zr and sl, the value at the midpoint of the line segment increases as sr decreases, and a point will arise at which the only way to fit a pair of quadratics smoothly along the line segment (assuming both sl and sr are negative) will be to force a turning point in each piece of quadratic and therefore a local maximum somewhere in the right hand half of the line segment. If the value of sr is very large (and negative) then there is no reason why this local maximum might not also be the global maximum in the piecewise quadratic along this line. Examination of our data supports the view that a two-dimensional manifestation of such behaviour was the cause of the phenomenon described above.

Reduction to a finer grid would appear to be the obvious solution to this problem; however it would have been necessary to increase the fineness of the grid quite considerably to achieve our objective, and such an alteration was clearly totally unnecessary in all other areas of the plot. It was therefore decided to take the alternative course of replacing the true gradients in the top right hand corner of all plots by estimated values. This of course is a rather ad hoc and unsatisfactory solution, though it was successful in eliminating the spurious peaks from all plots illustrated in this section. The possibility remains that less extreme forms of behaviour of this sort might well pass undetected by users and could provoke misinterpretation of contour maps produced by the piecewise quadratic method. The author is however of the firm belief that these examples are quite exceptional and that the phenomenon described above has not been observed in any other contour plots produced by the method. One potential means of eliminating the behaviour which we have described is to use a locally adaptive (non-uniform) grid to approximate our surface, and this idea is followed up in some depth in Chapter 6.

We now consider the (corrected) plot of our first data set, which comprises twenty pseudo-random Standard Normal observations from the NAG generator; these values are listed as data set no 1 in Table 4.1. In theory the function should be maximised on the dotted line $\mu = 0$, along which the likelihood is constant, but in practice we will only expect to observe an approximation to such behaviour, as a result of sampling error. In fact Figure 4.3 (which, in common with Figures 4.4 and 4.5, uses an 11 x 11 grid of points) suggests that the maximum occurs some distance from this line, indicating approximately a 15% contamination of observations with mean around -1.3 as the most likely cause of such a data set. This maximum is not very much greater than the value at $\mu = 0$, the surface being relatively flat over a large area of the plot, but it must be borne in mind that the log transform (base e) which we use has a distorting effect on the flatness of the surface. If the values listed as data set no 1 in Table 4.1 are plotted on graph paper there is indeed a strong suggestion of bimodality in the observations; it was therefore decided to examine a second set of pseudo-random Standard Normal observations produced by the NAG generator (listed as data set no 2 in Table 4.1).

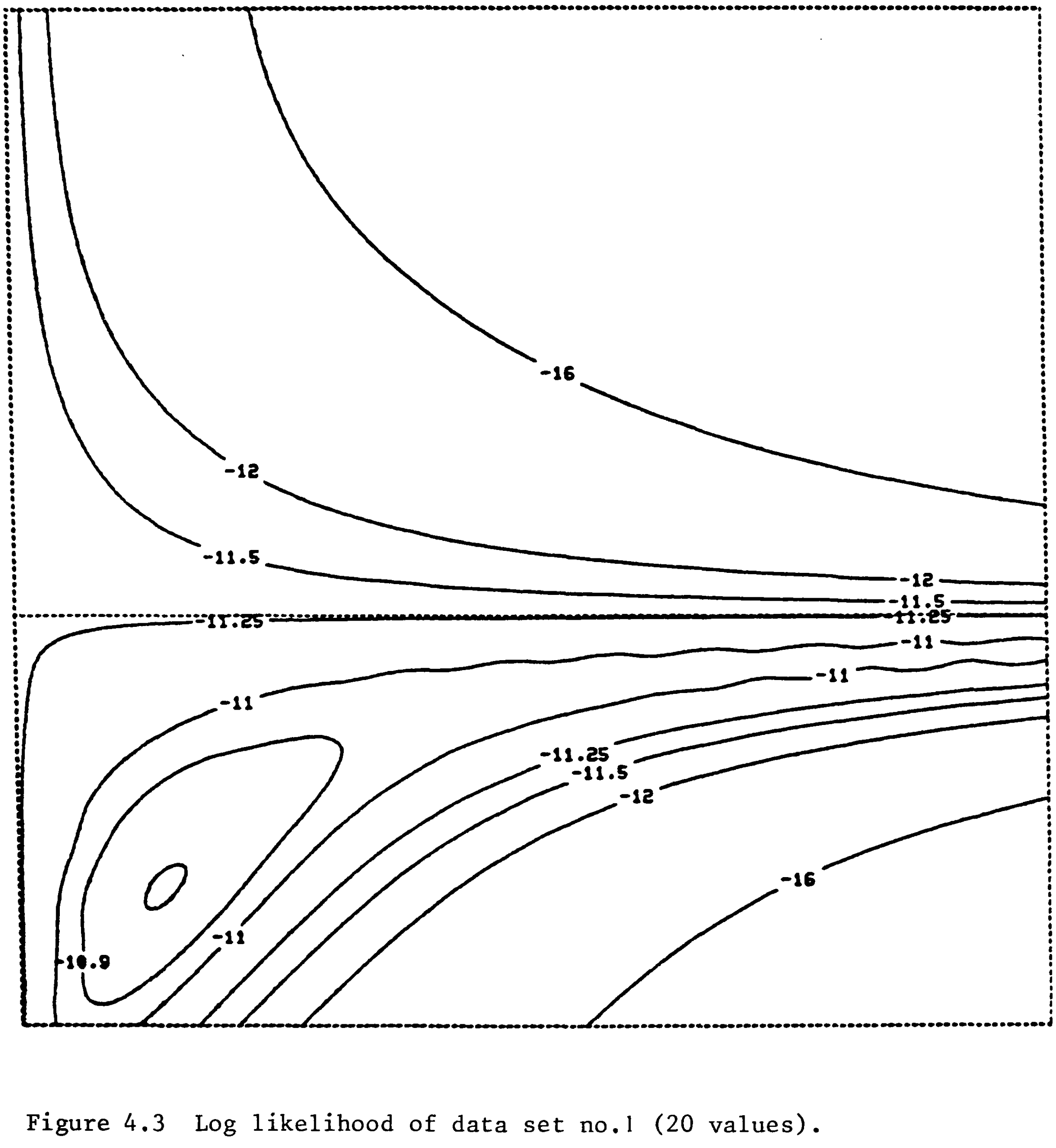


Figure 4.3 Log likelihood of data set no.1 (20 values).

Table 4.1

| Data set no. 1 | | Data set no. 2 | |
|---|---|---|---|
| -2.475 | 0.492 | 0.126 | 0.147 |
| 0.419 | 0.550 | -1.402 | -1.029 |
| 0.932 | 0.819 | -0.489 | -0.676 |
| 0.441 | 0.664 | 0.392 | 0.384 |
| -1.052 | -0.319 | -0.908 | -0.326 |
| 0.763 | -1.350 | 0.648 | -0.668 |
| 0.304 | 0.233 | 0.754 | 0.032 |
| -1.328 | -1.623 | 0.444 | -0.854 |
| -0.639 | -0.344 | -0.861 | -0.967 |
| -1.509 | 1.599 | -1.252 | 0.137 |

Figure 4.4 shows the likelihood plot resulting from the use of data set no 2. In this case there is no hint of bimodality in the data, but the data set is still not entirely satisfactory: both sample mean (-0.3184) and standard deviation (0.672) are disturbingly (though not quite significantly) lower than they should be. The maximum appears to occur at $\lambda = 1$, $\mu = \bar{y}$, but the surface is relatively flat whatever the mixing proportion for values of $\mu$ close to $\bar{y}$; other plots of pseudo-random Standard Normal variates indicate that the closer $\bar{y}$ is to 0, the flatter the surface becomes in the $\lambda$ direction for $\mu \simeq \bar{y}$. This is an intuitively obvious point: its interpretation is simply that the closer the sample mean is to zero, the more indifferent the model becomes to the value of the mixing proportion $\lambda$, with aliasing becoming total at $\mu = 0$.

Another property of the model, which is evident in all plots in this section, is that it tends to regard as equally likely a high mixing proportion with $\mu$ close to $\bar{y}$ on the one hand and a low mixing proportion with a more extreme value of $\mu$ on the other hand. This property

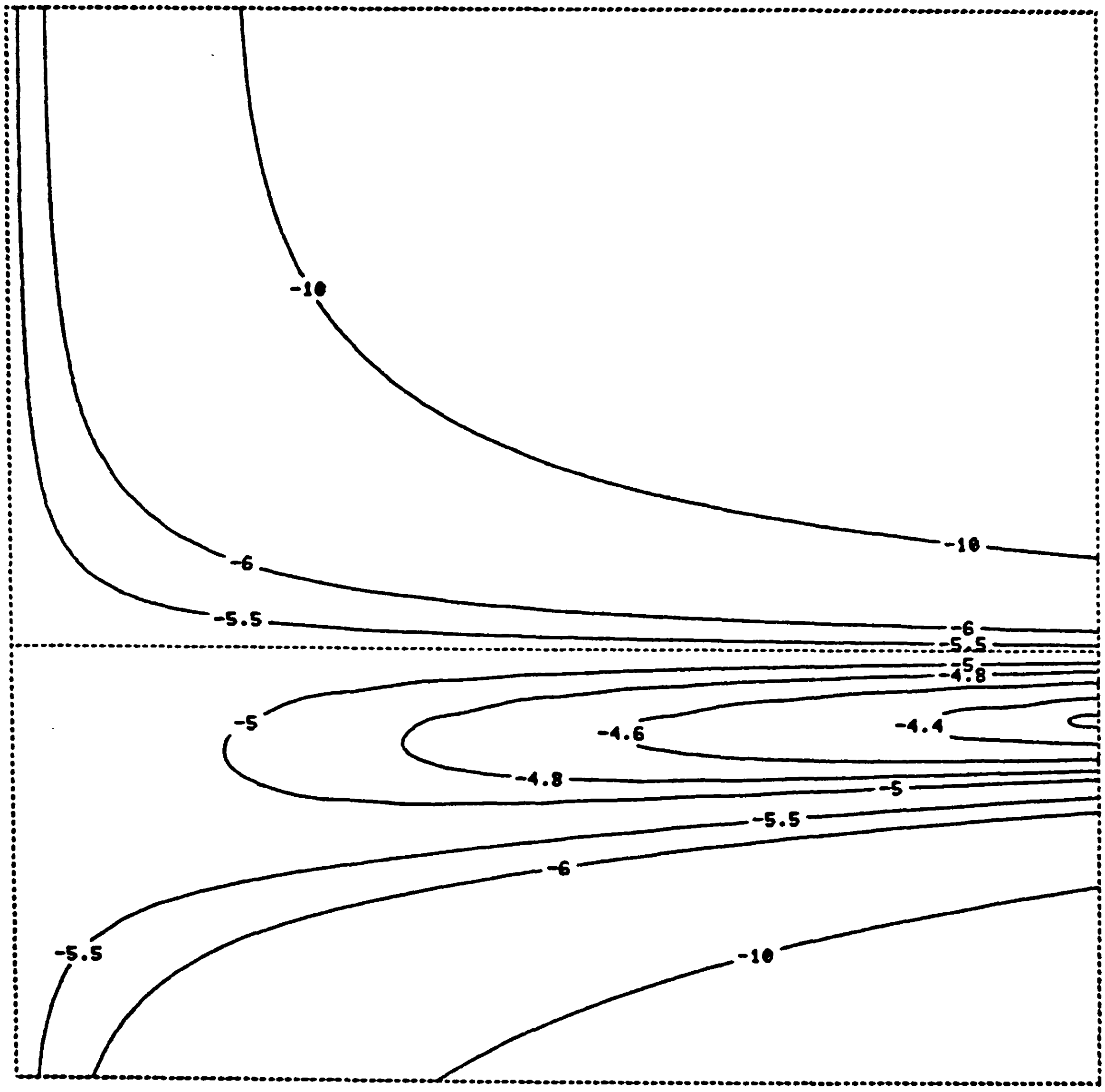


Figure 4.4 Log likelihood of data set no.2 (20 values).

manifests itself in the tendency of most contours to veer away from the line $\mu = 0$ as $\lambda$ decreases. However, this breaks down for the most likely sets of pairings of $\lambda$ and $\mu$, which tend to occur within more limited ranges of $\mu$ and (to a lesser extent) $\lambda$. Such cases are represented by contours which form single (rather than pairs of) branches lying on one side only of the line $\mu = 0$.

Although it is arguable whether data set no. 2 is really any more satisfactory than data set no. 1, the second data set was retained and transformed (by adding 2 to each value in the final column in Table 4.1) into a data set with values $\lambda = 0.5$, $\mu = 2.0$. The resultant likelihood function is contoured in Figure 4.5, the ideal maximum being indicated by a cross. It can be seen that the maximum occurs at a point where $\mu$ is barely greater than unity (though $\lambda$ is nearly correct), but this is hardly unexpected given the low mean of the original data set. The method has at least recognised that the data is probably bimodal.

In an attempt to eliminate some of the sampling error which has been so much in evidence thus far, we have simulated a much larger data set of 100 pseudo-random Standard Normal observations, to produce the likelihood function which is plotted as Figure 4.6. Even in this example the mean is rather low, but the likelihood function does appear to be settling down in its behaviour. This plot and the others in this section illustrate the intuitively obvious point that the method will always find the pairing $\lambda = 1$, $\mu = \bar{y}$ more likely than $\lambda = 0$, $\mu = 0$, no matter how close to Standard Normal the data are: this is a consequence of having to fix one of the means at zero in order that there should be no more than two parameters in the model.

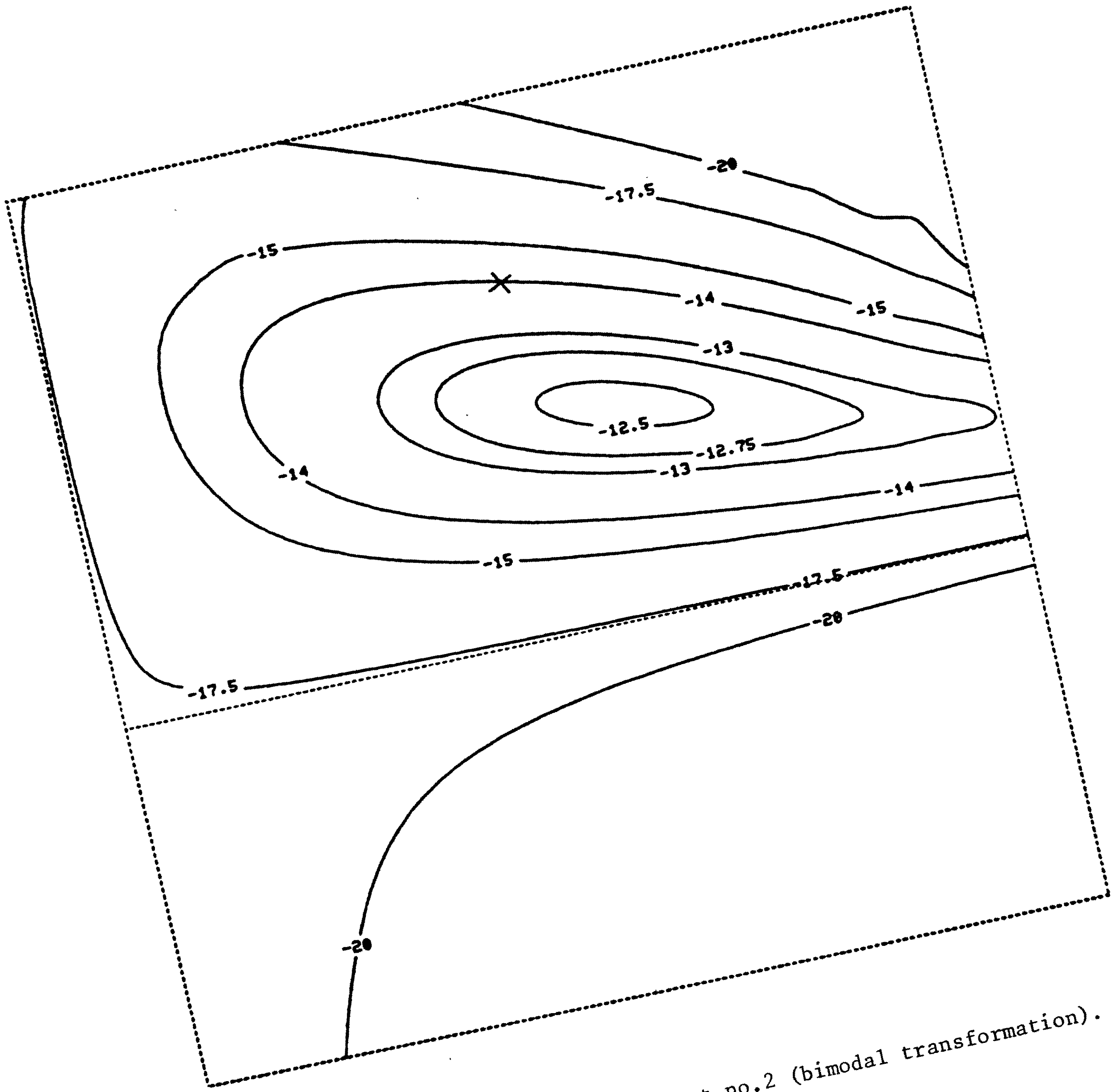


Figure 4.5 Log likelihood of data set no.2 (bimodal transformation).

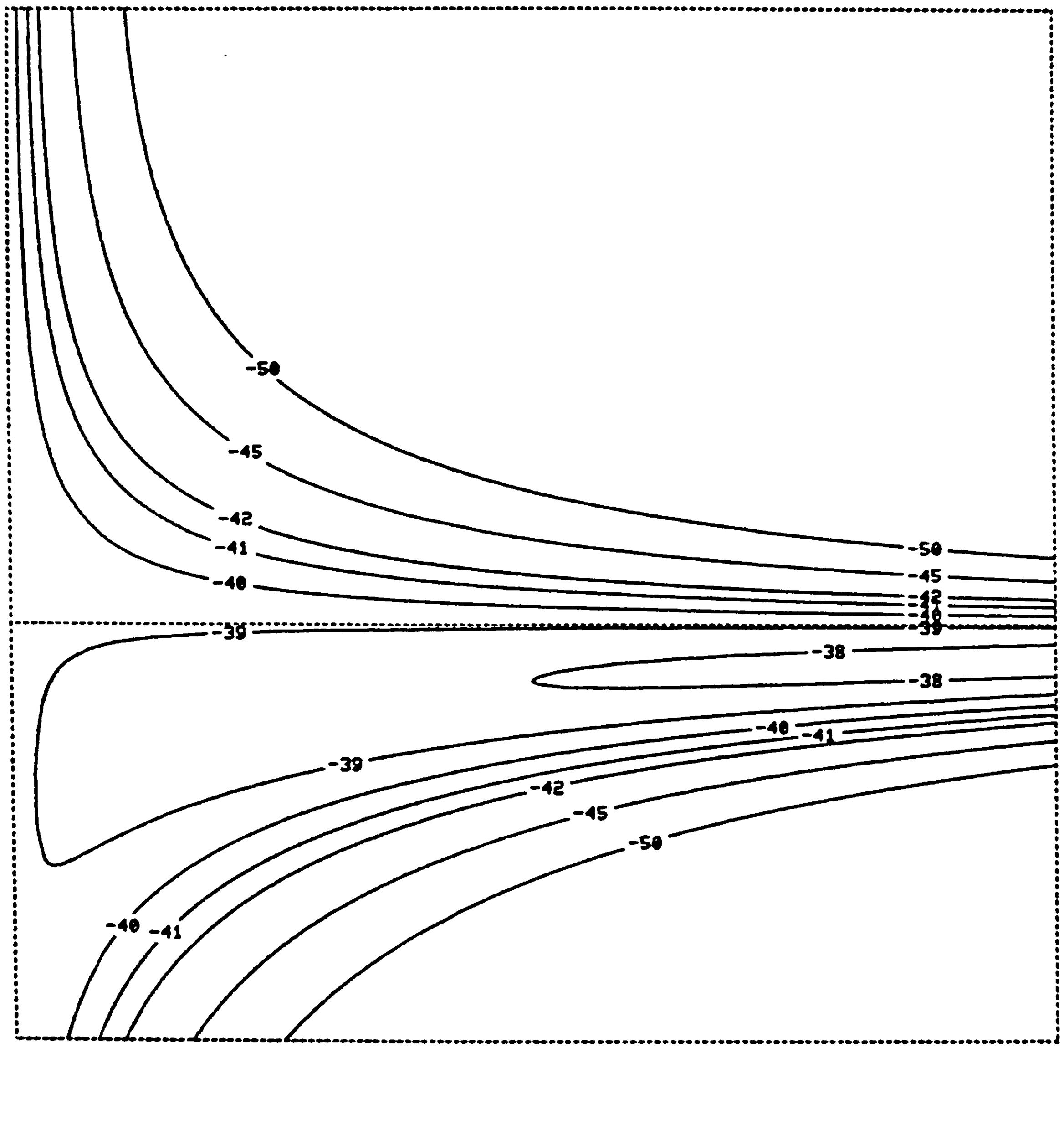


Figure 4.6  Log likelihood of 100-value data set.

Finally Figure 4.7 was produced by adding a value of 2 to half of the data used in the construction of Figure 4.6 (both plots use a 21 x 21 grid of values and gradients). It can be seen that the maximum is very much closer to the ideal maximum (again marked by a cross) than was the case in Figure 4.5.

It might be felt that we have been unlucky in this section in generating data sets of a fairly atypical nature. However it is important that we should not conveniently brush them aside in the hope of getting something better next time. The use of these data provides an important warning to the statistician that it is dangerous to make strong inferences from a small sample and provides an effective illustration of the phenomenon of sampling error.

However interpretation of the meaning of surfaces is a secondary consideration in this thesis. The most important conclusion which we draw from this section is that contouring by the piecewise quadratic method provides a highly effective means of studying the likelihood functions which arise from our model. A number of the details of surfaces which we have commented upon would almost certainly have passed unnoticed if an alternative method of presentation (such as the perspective block diagram) had been used.

## 4.4 Contouring meteorological data

Weather forecasting is undoubtedly one of the most important application areas for automatic contouring methods. The data which are contoured in this section were generated by the European Centre for Medium-range Weather Forecasts (ECMWF), and represent a selection of fairly typical surfaces which have arisen in global weather forecasting in recent years. The surface heights were provided already in gridded

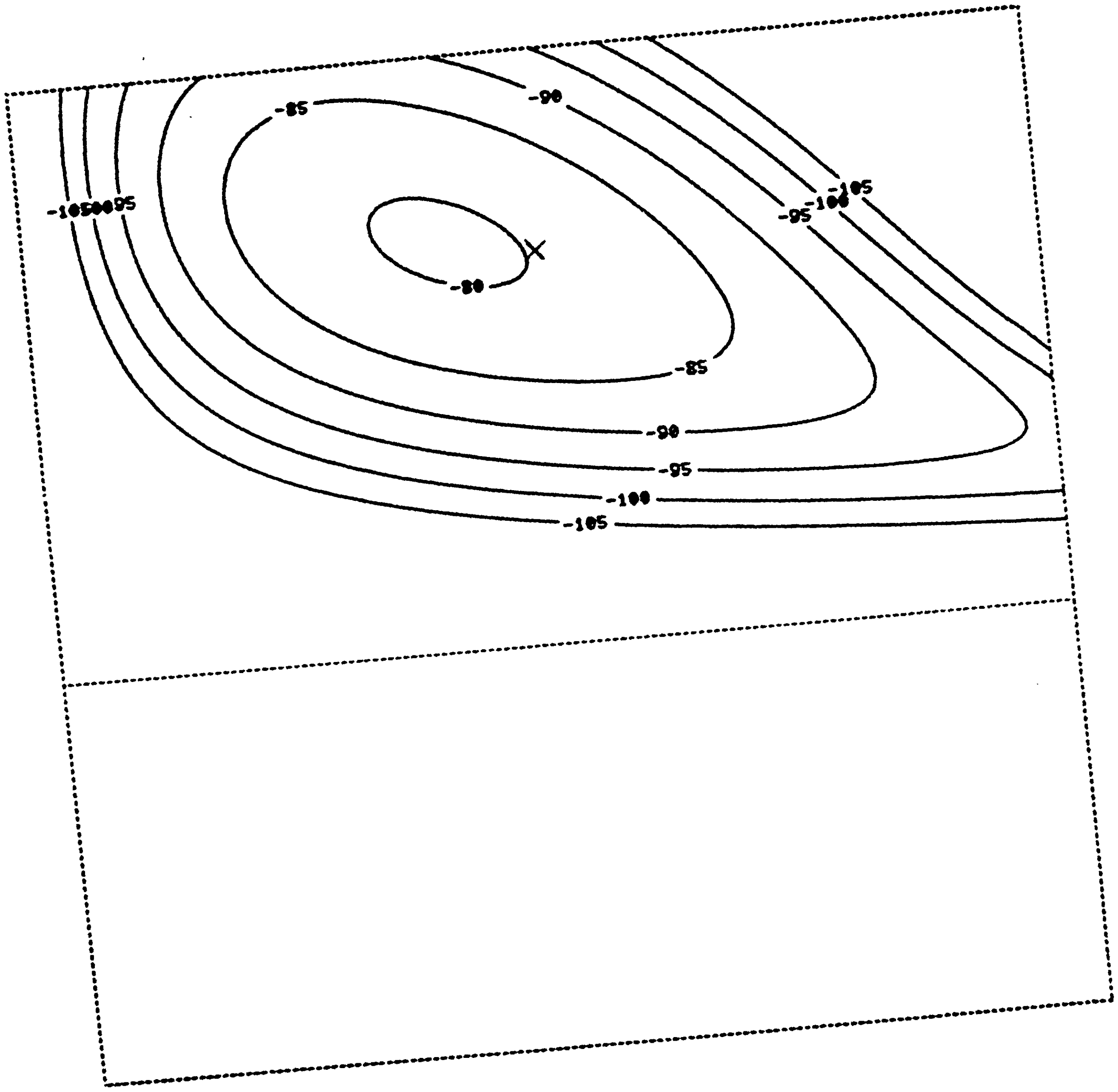


Figure 4.7  Log likelihood of 100-value data set (bimodal transformation).

form but gradient values were not available and have therefore been estimated in all examples. A distinctive feature of these data sets is their magnitude, the largest being a 240 x 61 grid of values. In most or all cases the grids were found to be unnecessarily large for contouring by CONICON, but were useful for the construction of accurate gradient estimates. Nevertheless the package succeeded in contouring all the complete data sets without error.

Unlike the illustrations elsewhere in this thesis, a number of the plots in this section were produced by an electrostatic plotter, following vector-to-raster conversion, at ECMWF. The size of these plots has been photographically reduced from 550mm square to 158mm square. The implementation of CONICON used at ECMWF incorporates a number of extra features which are not in the standard version of the package and are therefore not described in Chapter 3 or documented in Appendix A; most of these features are illustrated in one or more of the plots in this section.

We begin by investigating a 500mb geopotential field; that is the altitude (in tens of metres) at which atmospheric pressure equals 500 millibars. Fields of this type are often studied in preference to measurements of surface pressure because they are unaffected by changes in relief. A polar stereographic projection has been carried out on our data sites so that the area over which we are contouring, the Northern Hemisphere, appears flat and circular. The data were provided in the form of an 86 x 86 grid of heights, with all values falling outside the circle of interest being set to very large negative numbers.

Figure 4.8 is a plot of the complete data set, produced by the standard version of the package. As the circular area of our plot might have been approximated by a convex polygon, subroutine BORDER has been called to plot the boundary of the contoured area, though the values in ZLIM specifying where contouring should be suppressed were not in fact set by subroutine CONVEX.

It can be seen that the surface is a relatively simple one and there appears to be little justification for the use of such a fine grid. Figure 4.9 shows that this is true: in this illustration we have superimposed two plots of the surface, each of which uses a 29 x 29 subset of the original grid (every third point in each direction), the two subsets being mutually exclusive. It can be seen that over most of the plot the contours in each map are indistinguishable (and this is still true when the physical scale of the plot is increased by a factor of 2.8), and only in the flattest areas of the plot is there a slight visible difference. It should be noted that the full 86 x 86 grid of points was used in estimating the gradients in this example:- if the 29 x 29 grids alone had been used the discrepancies between the two surfaces would have been slightly greater.

Each of the 29 x 29 plots appearing in Figure 4.9 used approximately a third of the CPU time required to construct Figure 4.8; and in general we have found that for a given data set CPU usage tends to increase roughly in proportion with 1/grid size. There is a small quadratic term in the relationship but this is of little importance relative to the linear term; we infer from this that the bounds derived in Section 2.6 for values attained by the function within each element are accurate enough to allow us to discard almost all of the unwanted elements at any particular level without incurring significant costs.

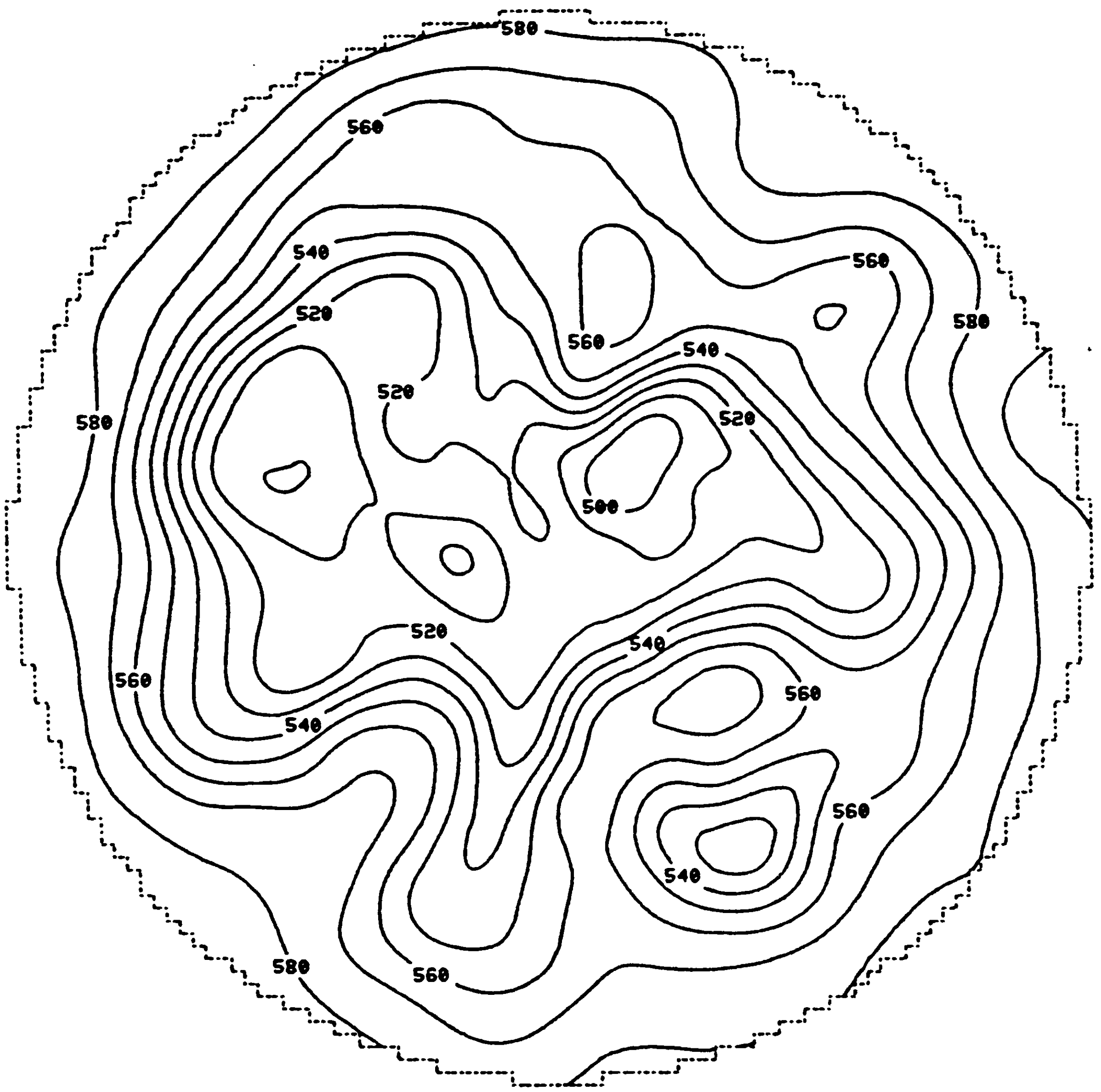


Figure 4.8 First ECMWF data set: 500 mb geopotential field (85 x 85 grid of elements, gradients estimated).

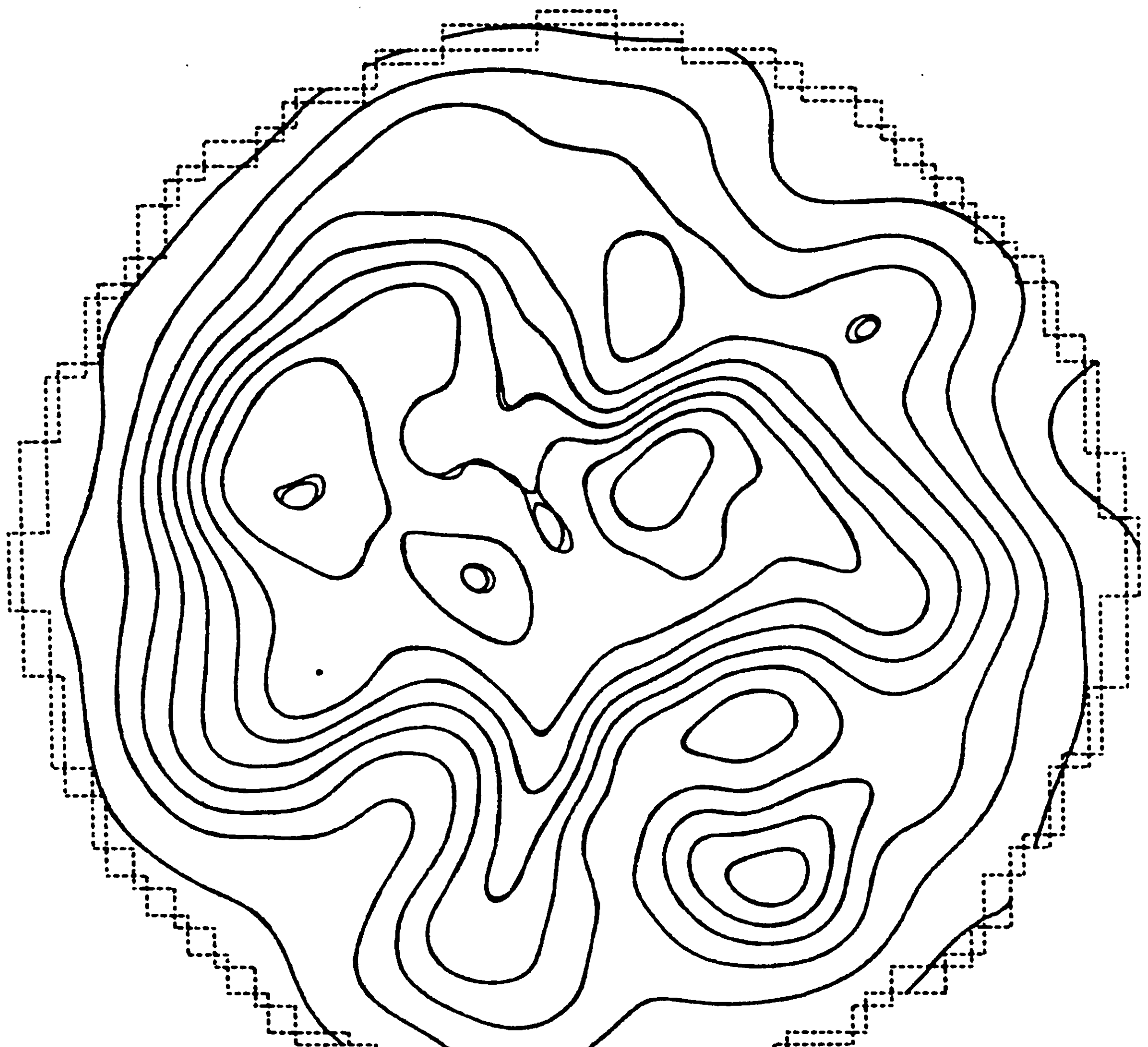


Figure 4.9 First ECMWF data set: two plots superimposed (each using a 28 x 28 grid of elements).

The reader may have noticed the policy adhered to in this thesis of not quoting absolute CPU timings; the major reason for this is a practical one - over the course of the project improvements have continually been made to different parts of the package which have reduced CPU usage and therefore rendered all previous CPU timings obsolete. Unfortunately constraints on time have prevented those jobs which created the illustrations in this thesis from being repeated, and therefore it has only been possible to provide an impression of relative times where it is felt that these might be of interest. A further problem arises in separating the CPU usage of the CONICON package itself from that of the basic graphics software provided by the user's system, which even in the case of the Avon Universities Honeywell Multics System implementation (which uses a highly efficient package of simple graphics routines) is believed to be responsible for 40% or more of total CPU usage. Thus when we do quote comparative CPU timings it should be borne in mind that these timings are inclusive of both the CONICON and basic graphics packages in the Avon Universities implementation, and it is therefore necessary to assess what proportion of the discrepancy can be explained by differences in the number of simple graphics instructions in each plot. It is clear that if a less efficient graphics package were interfaced to CONICON (and it is not uncommon to come across graphics packages which are several times less efficient than the package currently used) the basic graphics routines could dominate CPU usage to such an extent that the CONICON part of the job became relatively unimportant.

In the ECMWF implementation of CONICON CPU times have been compared with those relating to a piecewise linear package in regular use at that installation, which was written by staff at the Weather Centre (Petersen, 1978) and employs the algorithm of Dayhoff (1963). For a fixed grid size CONICON appears to use very approximately three times as much CPU time as the piecewise linear package, after CPU usage by the basic graphics software has been deducted. Now since Dayhoff's version of the piecewise linear method divides each grid cell into four triangles only we would expect CONICON to use more than twice as much CPU time as this package, even if it were to approximate each conic section by a single straight line segment and therefore omit calculation of conic parametrisations, the angle subtended by endpoints at the pole, etc. Moreover CONICON, unlike the piecewise linear package, has to perform an internal linking process and has not yet been compiled with the optimising compiler at the ECMWF installation (which can be expected to reduce CPU usage by up to 20%). It is therefore very unlikely that any gross inefficiencies remain in CONICON.

Of course it would be unfair to state without qualification that CONICON is three times slower than a piecewise linear package, because results produced by each package given a fixed grid size are so vastly different. As we stated early on in this thesis, in order to obtain results comparable in quality with those produced by CONICON using a piecewise linear method, grid size must be reduced to such an extent that contouring may become prohibitively expensive in terms of CPU time, memory requirements or the calculation of the grid values themselves.

We now present, in Figure 4.10, a single plot of a 29 x 29 subset of our data, on this occasion produced by the ECMWF version of the package. In this case, the coarse grid alone has been used for gradient estimation, but there is very little change in the appearance of the plot. Local minima and maxima of the surface have been plotted as 'L's and 'H's

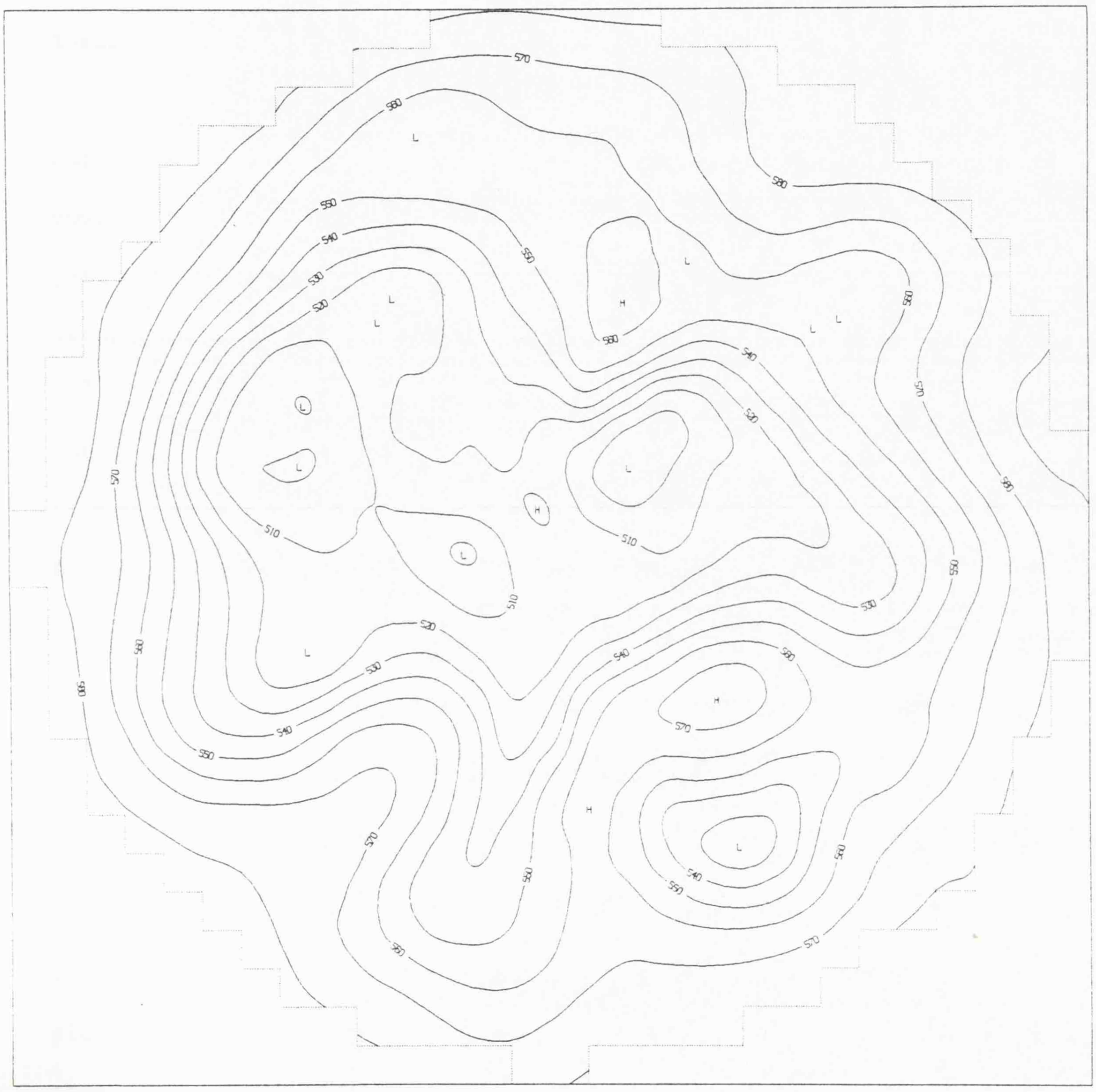


Figure 4.10 500 mb geopotential field contoured by ECMWF version of CONICON, with local maxima and minima indicated (28 x 28 grid of elements).

respectively (saddle points are not included), and one of the improvements made to the ECMWF implementation of the package is now apparent: labels have been plotted along contours in the same orientation as the contours themselves.

The number and positions of the 'high' and 'low' symbols in this particular plot would probably be quite acceptable to meteorologists; however, in plots of the same surface contoured over finer grids (in particular the full 86 x 86 grid of points) some extra stationary points occurred, resulting in some cases in the overlapping of pairs of 'H' and 'L' symbols. Checks showed that the extra stationary points were genuine; one such example is illustrated by Figure 4.11. Here we have a single element from the 86 x 86 grid in which a pair of minima (+ signs) and a pair of saddle points (0s) occur in very close proximity to one another. It is believed that such behaviour is a reflection of a suggestion in the gridded values that some higher derivative(s) may vanish at or near the cluster. This phenomenon would clearly be unacceptable to meteorologists and therefore a 'thinning out' process of some kind needs to be devised with their assistance. Such a process has been incorporated into ECMWF's piecewise linear contouring package - this requires the user to specify a radius of search in grid units; the package will then only plot local minima and maxima which are also extrema within a circle of that radius. To locate these extrema the package needs only to inspect values of the surface at grid points - clearly a more sophisticated approach would be required for use in conjunction with CONICON, where extreme values are not constrained to occur at nodes of the grid.

The final plot of this data set, Figure 4.12, shows the contours from Figure 4.10 crosshatched (using algorithm B) with a simulated

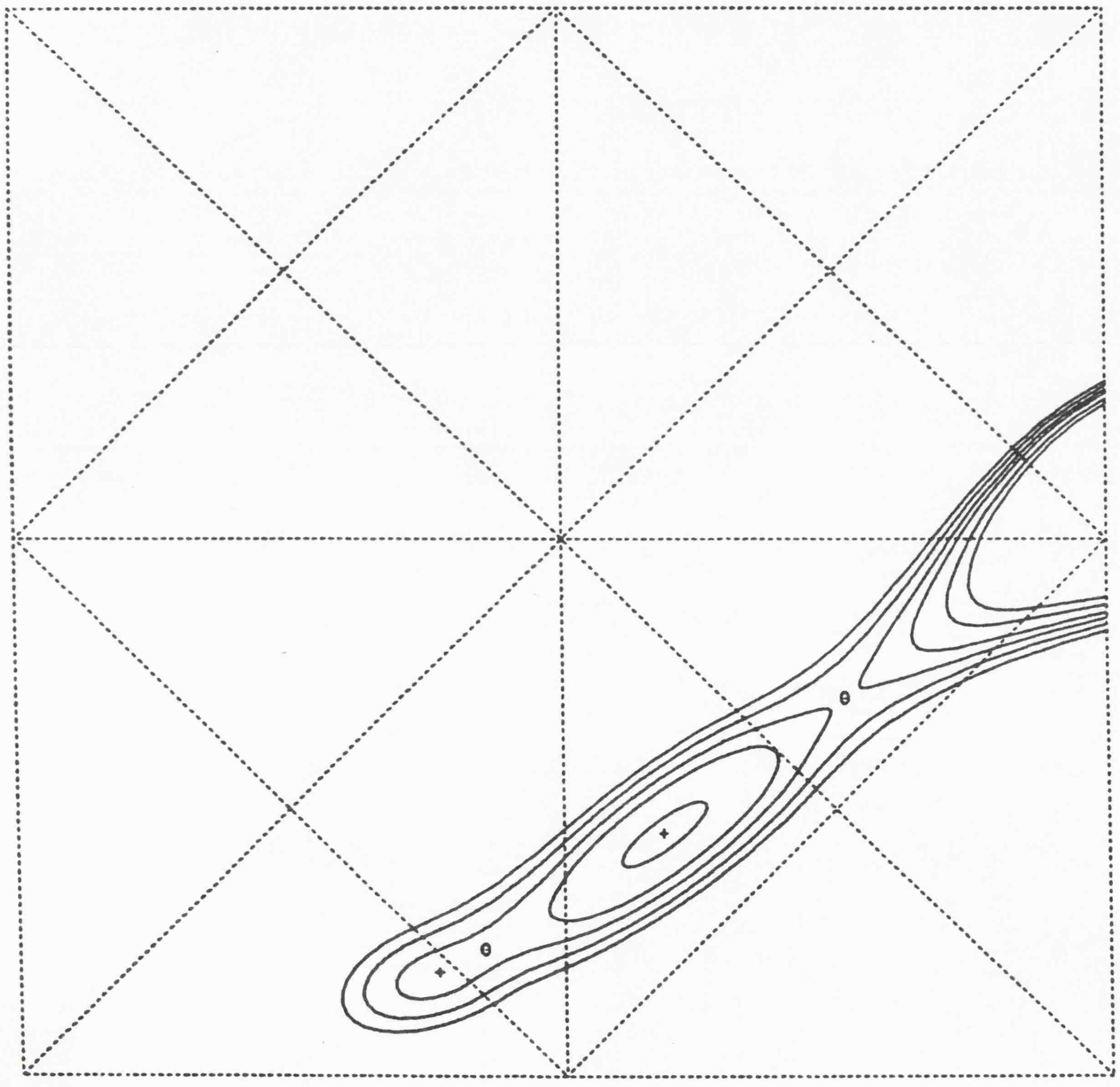


Figure 4.11  Clustering of stationary points within a single element.

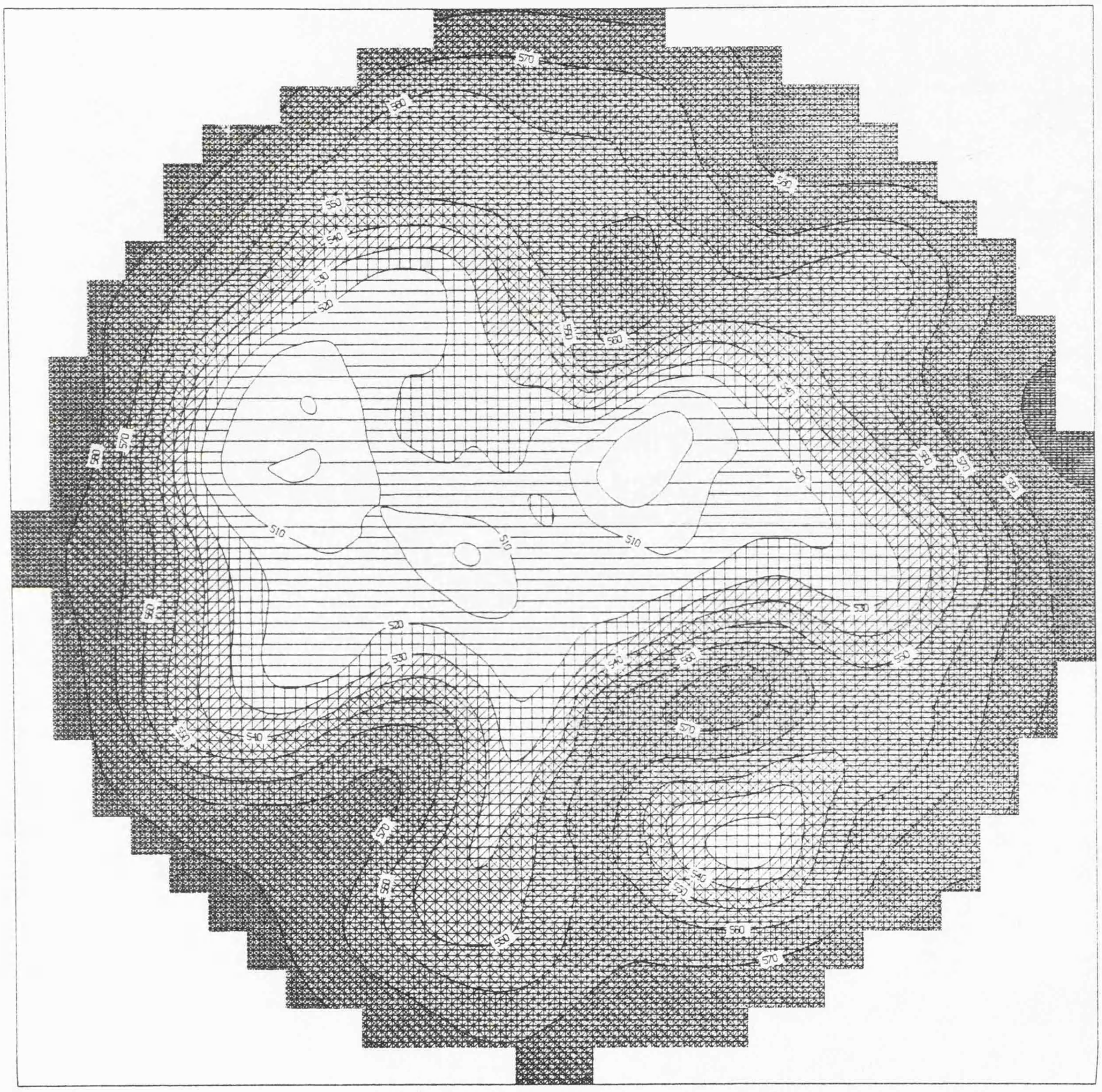


Figure 4.12 500 mb geopotential field contoured with crosshatching and annotation (28 x 28 grid of elements).

greyscale. Two more special features of the ECMWF implementation of CONICON are at once apparent:- crosshatching has been combined with the contour suppression feature (using an algorithm along the lines of that suggested in Chapter 3) and the construction of hatching lines has been modified to take account of the new labelling policy.

Unfortunately the nature of the variable being investigated in our second example is unknown; however we include plots of this data set to illustrate further features of the ECMWF implementation of CONICON. The data in this example have been projected onto a grid with a (nearly) square boundary, but in this case the elements are not square, the grid comprising 31 rows and 120 columns. The standard version of CONICON insists on the use of square elements, but as we have seen in Chapter 2 there is no theoretical reason why rectangular elements should not be used and the ECMWF implementation of the package has accordingly been adapted to contour a general rectangular grid. The number of alterations required to accommodate this useful extension was considerable, but all alterations were of a fairly trivial nature, one important reason for this being that the quadratic contouring routines employed in CONICON were designed to handle arbitrarily-shaped triangles.

Figure 4.13 illustrates how the ECMWF version of CONICON contoured this particular data set. It can be seen that none of the labels in this plot overlap each other or neighbouring contours, and this is not the result of a coincidence:- this version of the package includes a number of checks which will normally prevent labels from occurring in unsuitable positions.

Firstly, once the package (in particular subroutine LABEL) has calculated the intended position of a label it carries out a pair of checks in an attempt to ensure that the contour is relatively straight

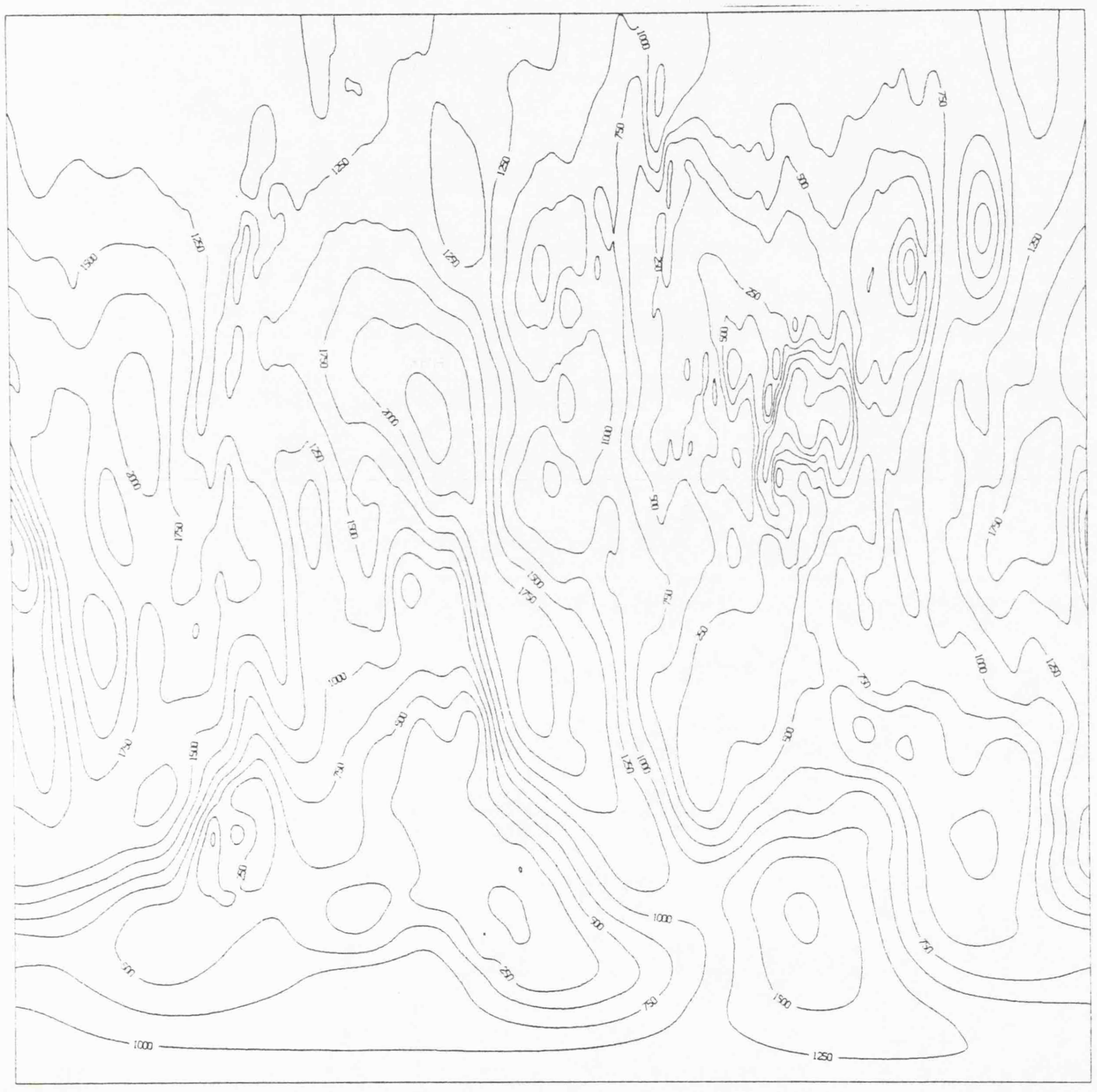


Figure 4.13 Second ECMWF data set (119 x 30 grid of rectangular elements).

in this area:- it begins by stepping along the contour from the centre of the label in both directions, until it locates the first point on the contour in each direction which is a distance greater than half the length of the label from the label's centre. It then calculates the angle subtended by these two points at the centre of the label and only allows the label to be plotted if this angle exceeds $150^{\circ}$. At the same time the slope of the straight line joining the two points just located is used to fix the orientation of the label. The second check considers the angle at which the contour enters the rectangular box around the label: if this is not within $30^{\circ}$ of a right angle the package again refuses to plot a label in this position.

However, if we pass both these tests and the package therefore considers the curvature of the contour in this area not to be excessive, we go on to examine the gradient of the surface in this area. This is done by locating the element in which the centre of the label is situated and considering the magnitude of the gradient at its centre and vertices; the maximum of these five values is used to summarize the gradient in this area of the plot. We are interested in the derivative in the direction normal to the orientation of the label, but this is also the gradient in the direction normal to the contour; that is, simply the magnitude of the gradient. We can therefore immediately compare our gradient value with the (vertical) distance to the nearest neighbouring contour level and decide whether there is sufficient room to fit the label (without obscuring any label which might exist on the neighbouring contour). If we fail any of these three tests the package immediately abandons its attempt to fit a label in the current position, and begins again at a point a little further along the contour.

The tests described above which consider the curvature of the contour at a proposed label site leave some opportunities for the occasional special case to escape detection, particularly if the surface is a very complicated one; however our tests are unlikely to fail often, and a foolproof method would probably be considerably more expensive. The gradient test has one obvious failing in that it cannot detect the close proximity of a contour at the same level; one example where this has led to a contour almost touching a label is evident at the 250 level in Figure 4.13. This test may also fail to detect the odd special case, but nevertheless it is in general effective and very cheap. The alternative method of storing all contours before plotting them and then taking elaborate precautions to ensure that all contours respect the positions of all labels and no two labels overlap has been implemented in some packages (for example see the contouring feature of the DISSPLA graphics package (ISSCO, 1982)), but in the author's experience it tends to be excessively expensive both in terms of CPU and memory usage. Moreover it is often difficult to identify to which contour a label belongs in areas of high gradient. The alterations made to the ECMWF version of CONICON regarding positioning of labels make no significant difference to the overall cost of producing a contour plot and are in general highly effective.

Returning once again to our second data set, Figure 4.14 was produced by removing three out of every four columns from the original grid, leaving a 29 x 30 grid of square elements. Values on the coarser grid alone were used for gradient estimation and the combination of sparser data and poorer gradient estimates has in this case undeniably resulted in the loss of a large amount of detail from the contours. In In this example the complexity of the surface does appear to warrant

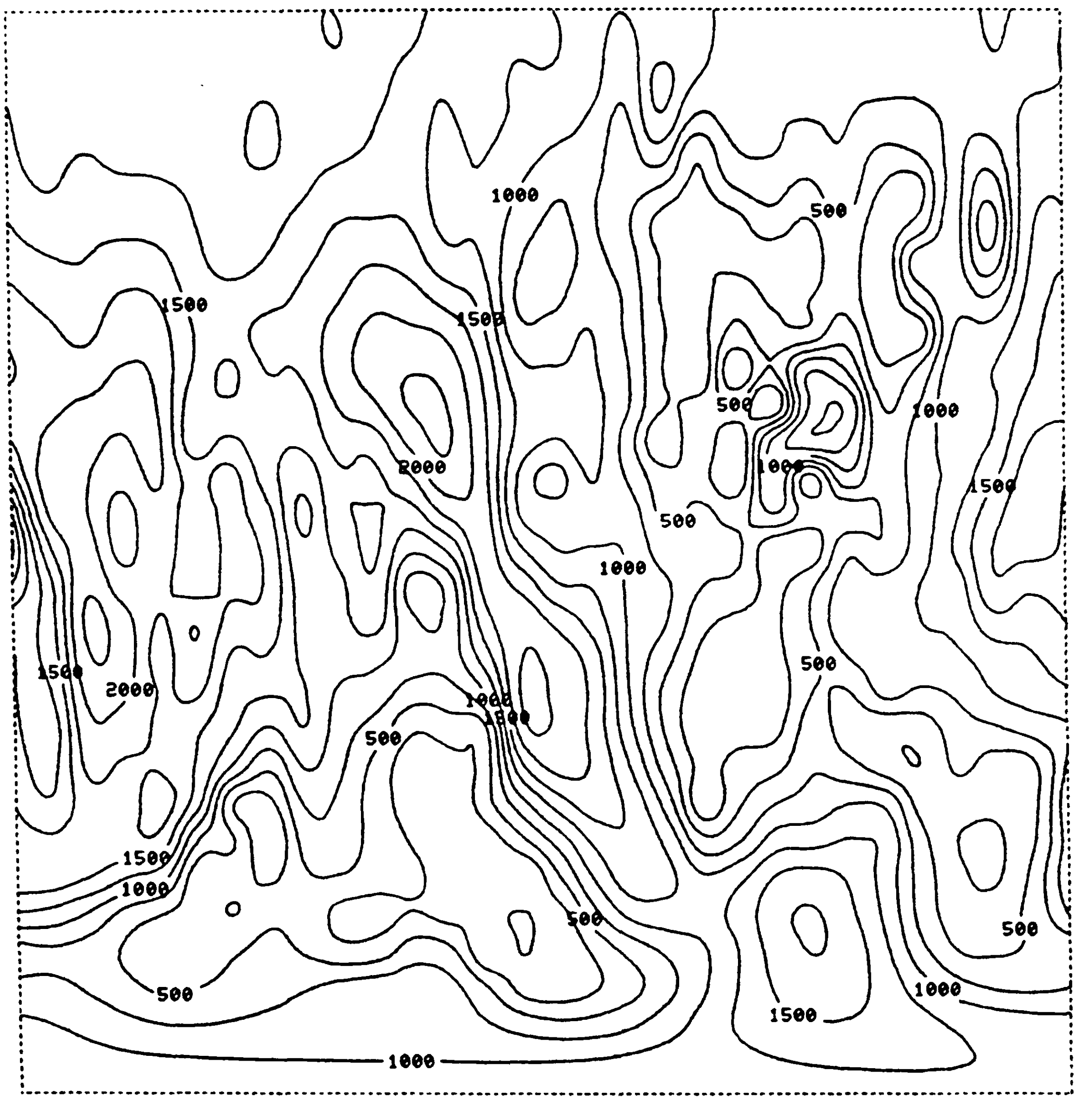


Figure 4.14 Second ECMWF data set, contoured by the standard version of CONICON. (29 x 30 grid of square elements.)

the use of a relatively large data set, even if only for gradient estimation. In order to compare the standard and improved labelling policies directly we have on this occasion illustrated a plot produced by the standard version of the package: useful comparison is however impaired by the use of characters of different sizes.

Figure 4.15 shows how CONICON handles the largest data set which it has thus far had to contour, a 240 x 61 grid of values representing relative humidity (expressed as a percentage). The plot is indeed complex, but there can be no justification for the use of 240 columns in the grid: even in the largest-scale plot which the graphics device can produce (a 550mm square) elements would be less than 2½mm across in the horizontal direction. Reduction to a 60 x 61 grid of square elements has very little effect on the appearance of the plot (once again CPU usage falls by about 50%), particularly if the full grid is used for gradient estimation.

However we have retained the full grid for the production of Figure 4.16, a crosshatched version, in order to illustrate that crosshatching algorithm B copes easily with a data set of this magnitude. In this and other plots (produced by algorithm B) in which crosshatching is used to build up density in this way, the crosshatching part of the job tends to be less time-consuming than the contouring part:- in a typical plot crosshatching might involve around 35% of the total CPU usage.

Finally we note that in a few areas of these plots the surface has exceeded the 100 level, behaviour which is of course impossible in practice - this is not a fault of the method, but rather reflects the fact that the surface is not wholly suitable for contouring by any general contouring method. Problems of a similar nature but greater

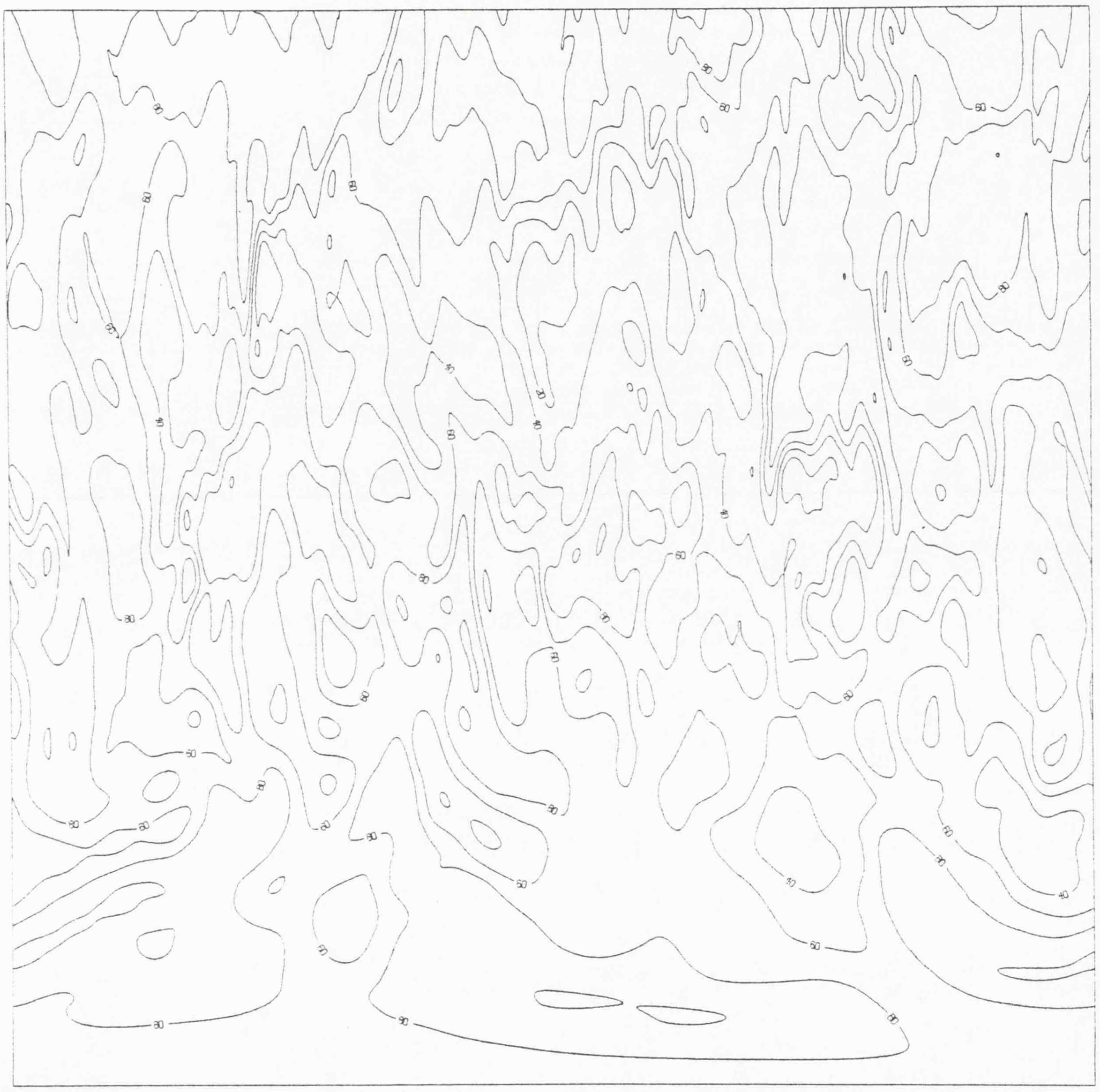


Figure 4.15 Third ECMWF data set: relative humidity (239 x 60 grid of rectangular elements).

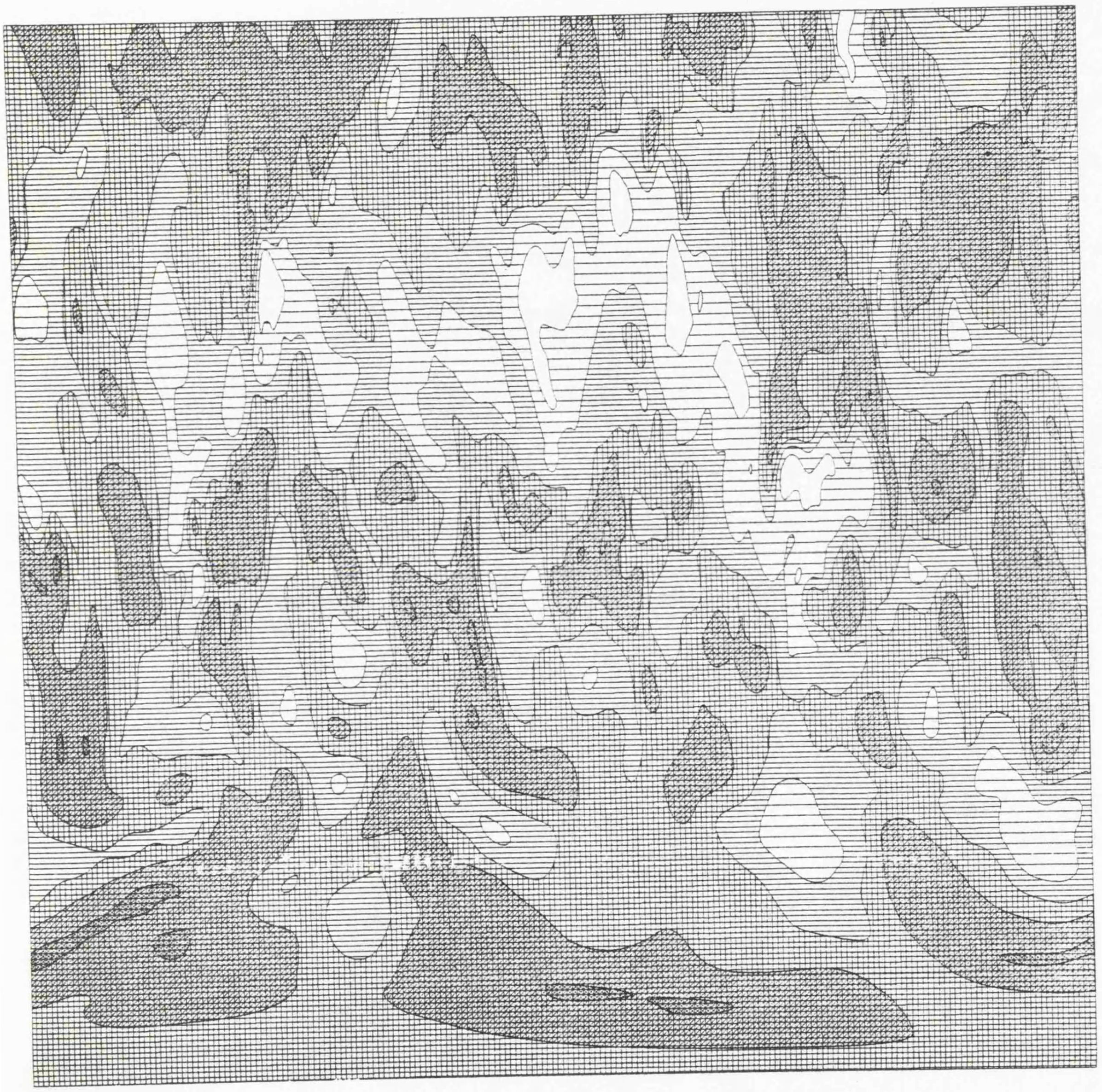


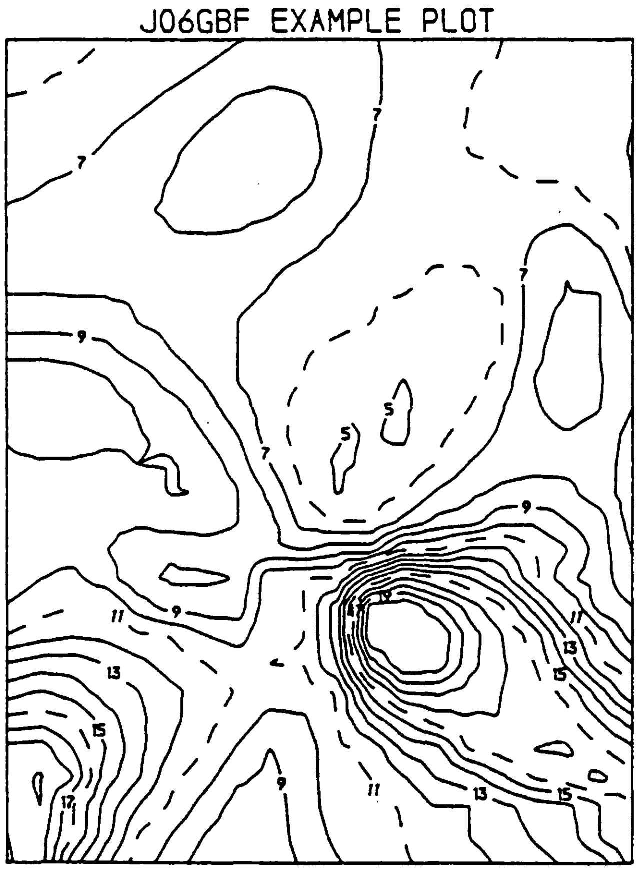
Figure 4.16 Relative humidity plot, crosshatched (239 x 60 grid of elements).

severity can arise if we attempt to contour rainfall data which may have extensive flat patches where the surface is zero. If we attempt to identify the dry areas by contouring very close to the zero level then breakdown of the Implicit Function Theorem can lead to the sort of anomalous behaviour in contours which was documented in Chapter 2 of this thesis. However, so long as we do not venture too close to the zero level, CONICON is perfectly capable of contouring rainfall data and has done so successfully on a number of occasions.

## 4.5 Plots of published data

In this final section we use the piecewise quadratic method to map three data sets which are contoured by alternative methods in the literature, and for the purpose of comparison we reproduce (with permission) the relevant illustrations from published sources. When comparing plots of the second and third surfaces it should be borne in mind that differences are not purely a consequence of using different contouring techniques - in these examples data sites are scattered irregularly and the methods of interpolation used also differ.

However we begin with a relatively simple example in which the data sites form a 30 x 20 square grid of points. These data were taken from the NAG Graphical Supplement (Numerical Algorithms Group, 1981), which gives no indication of the nature of the variable being measured. The NAG Graphical Supplement uses the piecewise linear method of Heap and Pink (1969) to construct contours, but also provides optionally a choice of two curve-fitting algorithms to increase their visual smoothness. Figure 4.17 shows the effect of selecting the piecewise cubic method of Butland (1980) to smooth contours of the surface. This may be compared with the CONICON plot of the same data set (with gradients



| CONTOUR | KEY |
|---|---|
| 1 | -200.0 |
| 2 | -180.0 |
| 3 | -160.0 |
| 4 | -140.0 |
| 5 | -120.0 |
| 6 | -100.0 |
| 7 | -80.0 |
| 8 | -60.0 |
| 9 | -40.0 |
| 10 | -20.0 |
| 11 | 0.0 |
| 12 | 20.0 |
| 13 | 40.0 |
| 14 | 60.0 |
| 15 | 80.0 |
| 16 | 100.0 |
| 17 | 120.0 |
| 18 | 140.0 |
| 19 | 160.0 |
| 20 | 180.0 |

Figure 4.17 NAG Graphical Supplement contour algorithm example plot (29 x 19 grid of cells. Reproduced by permission of Numerical Algorithms Group).

estimated by subroutine GRSET) which we present as Figure 4.18. The latter plot has been rotated through $90^o$ to allow a larger scale.

Comparisons between the plots show that even following the application of a curve-fitting algorithm to the contours defined by the piecewise linear method (with the consequent risks of neighbouring contours crossing each other), contours are still considerably less smooth than those produced by the piecewise quadratic method. In addition, one of the contours in Figure 4.17 touches itself in a manner which, though possible, is extremely unlikely and is probably an artifact of the smoothing technique. It is therefore felt that Figure 4.18 provides a much more convincing representation of the surface than Figure 4.17.

It was stated in Chapter 1 that we regard the two processes of contouring and interpolation as quite separate. However in some cases in the literature authors present a single method which combines the two processes (for example Powell and Sabin, 1977), and in many other cases the distinction between the two processes is hazy. An example of the latter type is Schagen (1982), who suggests using a method of the contour-following type in combination with his own two-dimensional interpolation method (Schagen, 1979). Ostensibly we have two separate processes here; however Schagen's contouring method relies on knowledge of the original data sites to enable itself to locate contours and cannot therefore be separated entirely from the interpolation process without alteration.

Schagen presents a pair of contour plots arising from a set of 72 measured permeability values from oilwells in a Russian oilfield, the Shkapovskii oil deposit, which we reproduce here (with permission) as Figure 4.19. The first plot was produced by the GPCP package (CALCOMP Inc, 1971) and the second by LUCAS, an implementation of his own

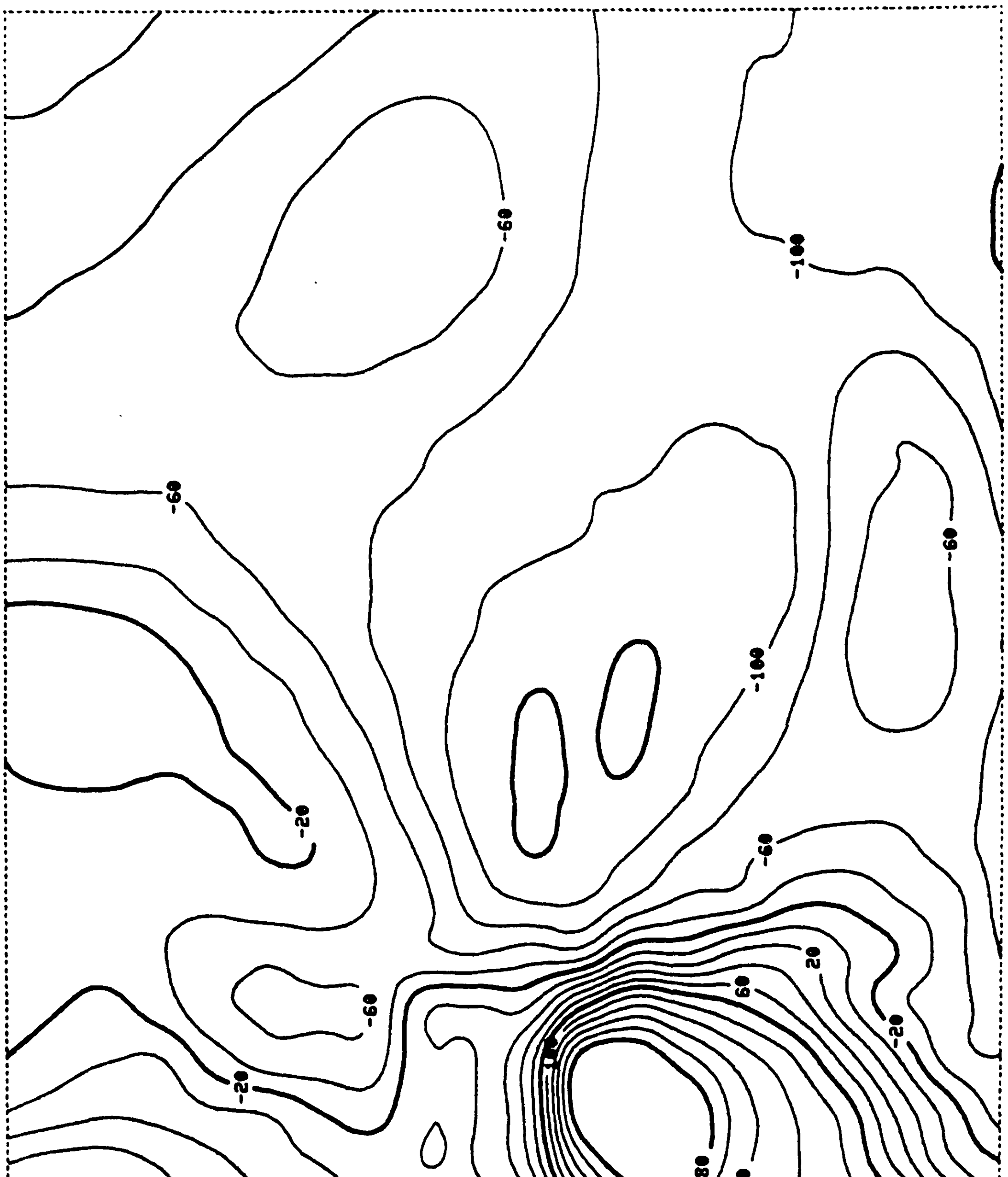
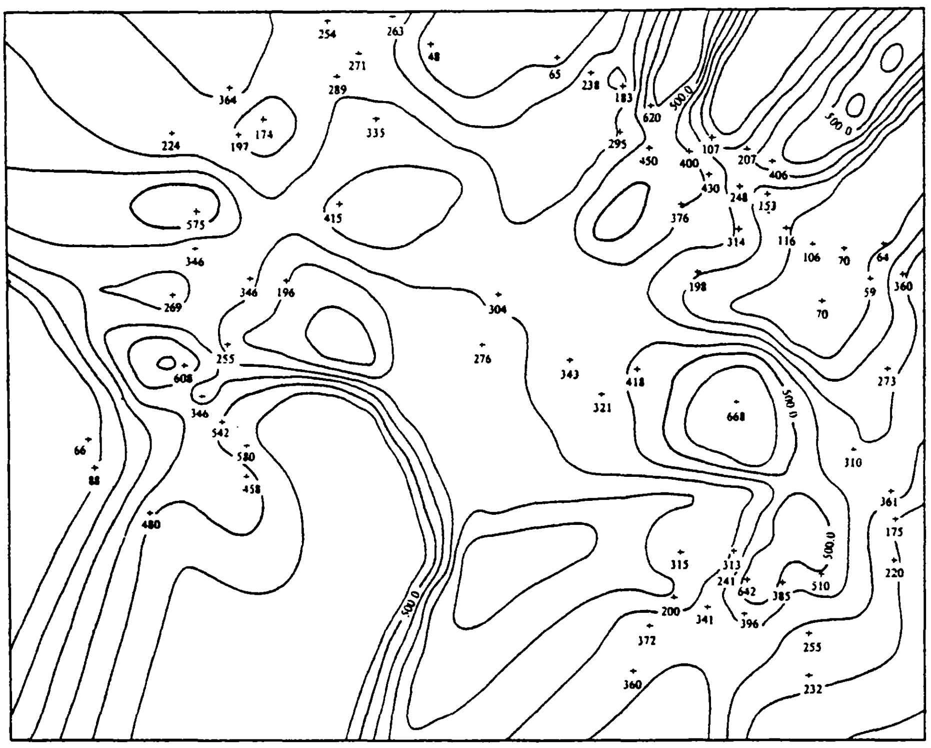


Figure 4.18 NAG example contoured by CONICON (29 x 19 grid of elements).



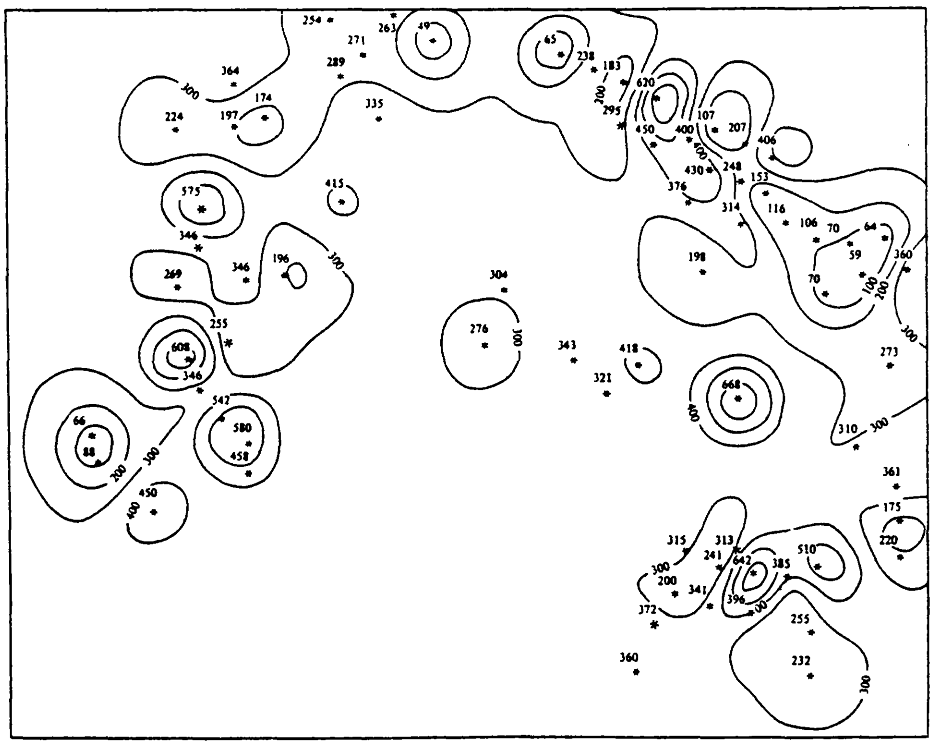


Figure 4.19 Shkapovskii data contoured by (a) GPCP and (b) Schagen's method. (Reproduced from the Computer Journal by permission of John Wiley and Sons.)

method. Comparison of these plots shows the strikingly different interpretations of the underlying surface which the two methods have provided.

Since the GPCP package employs what is essentially a piecewise linear contouring method and an interpolation method which is fundamentally different from Schagen's we find ourselves at the same time comparing different contouring and interpolation methods. Both contouring methods have succeeded in producing smooth curves (probably at considerable expense), though as we shall see this does not mean that the contouring techniques have both performed satisfactorily.

Not surprisingly Schagen argues that the plot created by his own method is the more plausible of the two; he points out that GPCP has produced some unlikely and apparently unjustified features which include a steep cliff-edge in the middle of a large area where there are no data sites, and that the contours in the GPCP plot are not always consistent with the original data. Schagen's criticisms of GPCP seem well founded, but his arguments in favour of his own method are much less convincing. He attempts to justify the conspicuous absence of contours in areas where there are no data sites by saying that this is an indication of "unknown territory" - a rather startling claim when one considers that the very purpose of an interpolation method is to attempt to provide an explanation of what might be happening in such areas.

However confusion arises at this point because it is not perfectly clear that Schagen's interpolant really is flat in these areas. His interpolation method considers the data to be a realisation of a stationary correlated random process in the plane; it is therefore possible that in parts of the plot which are a long distance from the nearest data site correlations are so low that the mean parameter becomes dominant - this would result in flatness even if there was evidence to support some other form of behaviour. However we cannot be certain that this is happening because any peaks which do occur in the interpolant in such areas would not in any case have been plotted by the contouring technique. Schagen's contouring method can only plot what he calls 'definable' contour segments - that is contour segments which divide the area of interest into two parts, each containing at least one data point - and will therefore not plot a closed loop unless it contains at least one data site. This is quite evident in Figure 4.19, though the appearance of a pair of ring contours which contain no data sites is now a mystery. This suggests that Schagen may have added heuristics to his algorithm to help identify contour segments which would not otherwise be 'definable'. If this is true then it seems likely that the interpolant is indeed flat in the central area of the plot.

Another disturbing feature of Schagen's map is that an inordinately large number of data sites occur at or very close to local minima and maxima of the surface, suggesting that the surface is much 'rougher' than it needs to be. The interpolation method is probably largely responsible for this behaviour, but the fault in the contouring method which we have discussed reduces the number of stationary points identifiable in the plot and therefore makes this property even more noticeable.

We note also that it is not at all clear that Schagen has solved the problems associated with contour-following methods which were mentioned in Chapter 1: he does not state how the method recognises when a closed loop has been completed; nor does he explain adequately his algorithm for locating contours which, in any case, as we have seen, is imperfect. The algorithm is based on a system of 'reference points' (intersections of contours with straight lines linking pairs of data sites), but the way in which this system is constructed is barely touched upon.

The data which Schagen contours originate from a paper by Schvidler (1964), but in neither publication are the coordinates of the data sites tabulated: they are simply plotted as symbols on a map. Thus a rather inaccurate digitisation process had to be carried out before the $C^1$ Natural Neighbour method of Interpolation (Sibson, 1982) was applied to the data to construct values and gradients on a 36 x 29 grid of points. This grid was then contoured using CONICON, resulting in the plot which we present as Figure 4.20. As we know that the piecewise quadratic method can be relied upon to produce smooth and accurate contours we are carrying out largely a comparison of interpolants when we compare this illustration with those in Figure 4.19.

The Natural Neighbour Interpolant of these data has features in common with both interpolants contoured in Schagen's paper and indeed appears to some extent to be a compromise between the two plots:- in the large central area which is free of any data sites it exhibits neither of the extremes which occur in the other two plots, but instead makes what appears to be a plausible guess on the basis of the available evidence as to the behaviour of the surface in this area. In common with the GPCP interpolant there are cases (though no more than two or three) where contours are inconsistent with data values (a property of the contouring method - not of Natural Neighbour Interpolation), but none of the spurious near-cliff edges

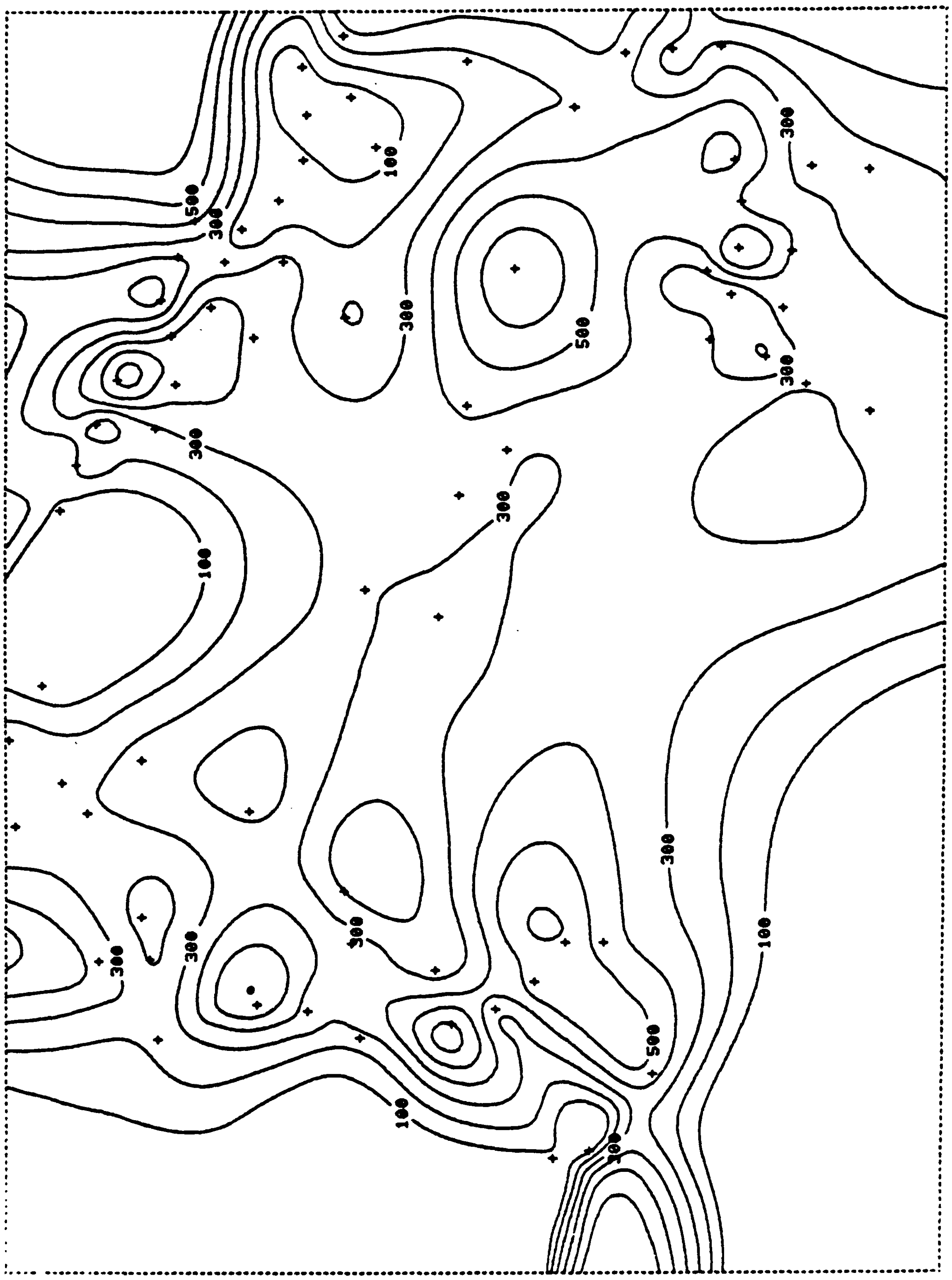


Figure 4.20 Shkapovskii data interpolated by the $C^1$ Natural Neighbour method and contoured by CONICON (35 x 28 grid of elements).

which occur in the GPCP interpolant are present in this plot; and it is no longer the rule that we find a data site somewhere within each closed loop contour. For these reasons it is believed that the Natural Neighbour Interpolant is by far the most plausible of the three. This example provides us with further justification for the use of the Natural Neighbour method in examples in this thesis which require interpolation as well as contouring.

The differences between interpolation methods which we encounter above afford us few opportunities to compare contouring techniques, but the following example presents us with a much better chance to do this. Once again we have a set of scattered data, 190 values representing a geological variable - elevation of the top of the Lansing Group (Pennsylvanian) in a part of Graham County, Kansas. These data were taken from the user's guide for the Surface II Graphics System (Sampson, 1975, revised 1978) and are used extensively in that publication to illustrate features of the package. Like most other contouring packages, Surface II Graphics employs a piecewise linear method of approximation and this is immediately evident from Figure 4.21 which illustrates a pair of typical plots of our data set. Even in the lower plot, which employs a 101 x 61 grid of values, a considerable amount of angularity is apparent in areas of high curvature.

In Figure 4.22 we have interpolated the data by means of the $C^1$ Natural Neighbour Method to give values and gradients on a 21 x 13 square grid of points, and plotted the contours using the piecewise quadratic method. Refinement to a 40 x 24 grid of elements makes little difference to the appearance of the map. In this case the two interpolants which we are comparing are very similar in

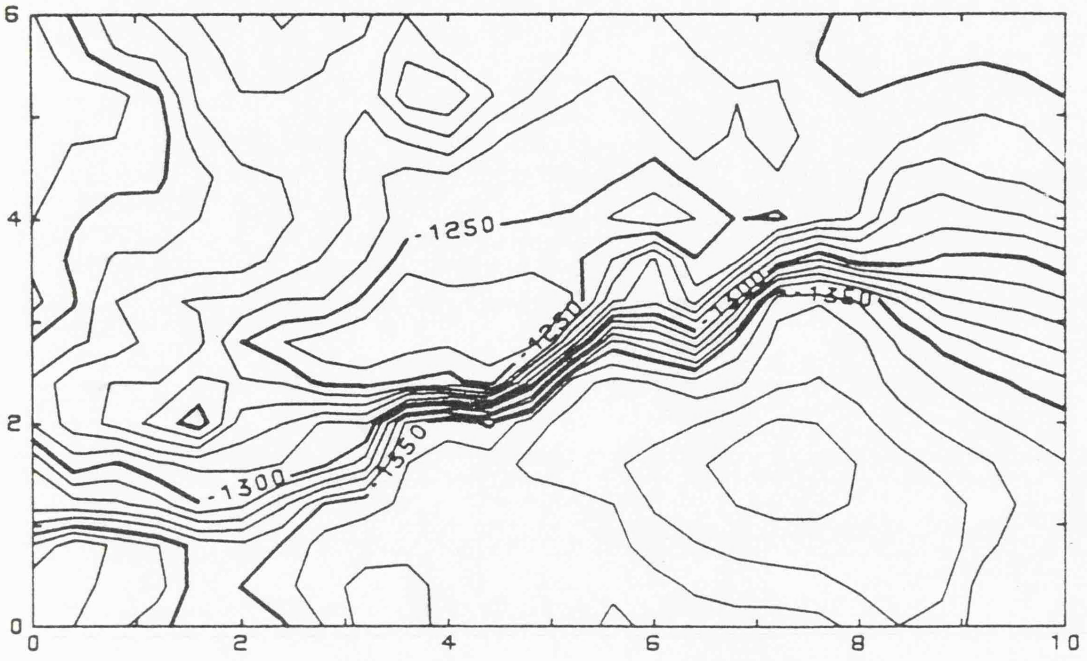


Figure 76.--Contour map showing subsurface structural elevation of the top of the Lansing Group in part of Graham Co., Kansas. Grid matrix contains 16 rows and 26 columns.

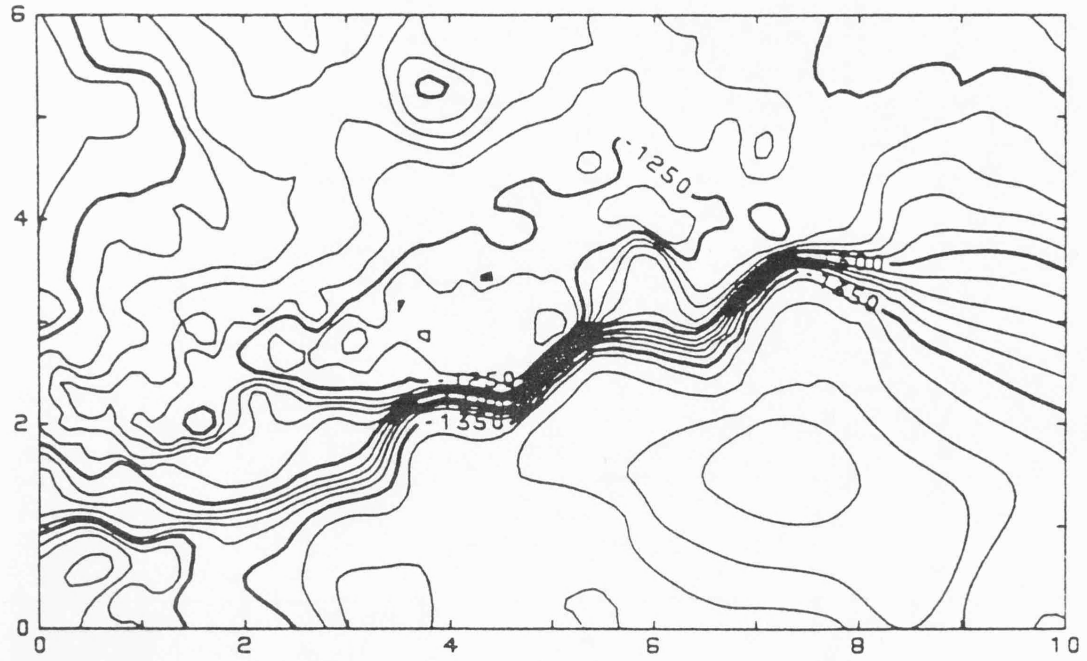


Figure 77.--Contour map of data from Figure 76, gridded with 61 rows and 101 columns.

Figure 4.21 Surface II Graphics System example plots (Reproduced by permission of Kansas Geological Survey).

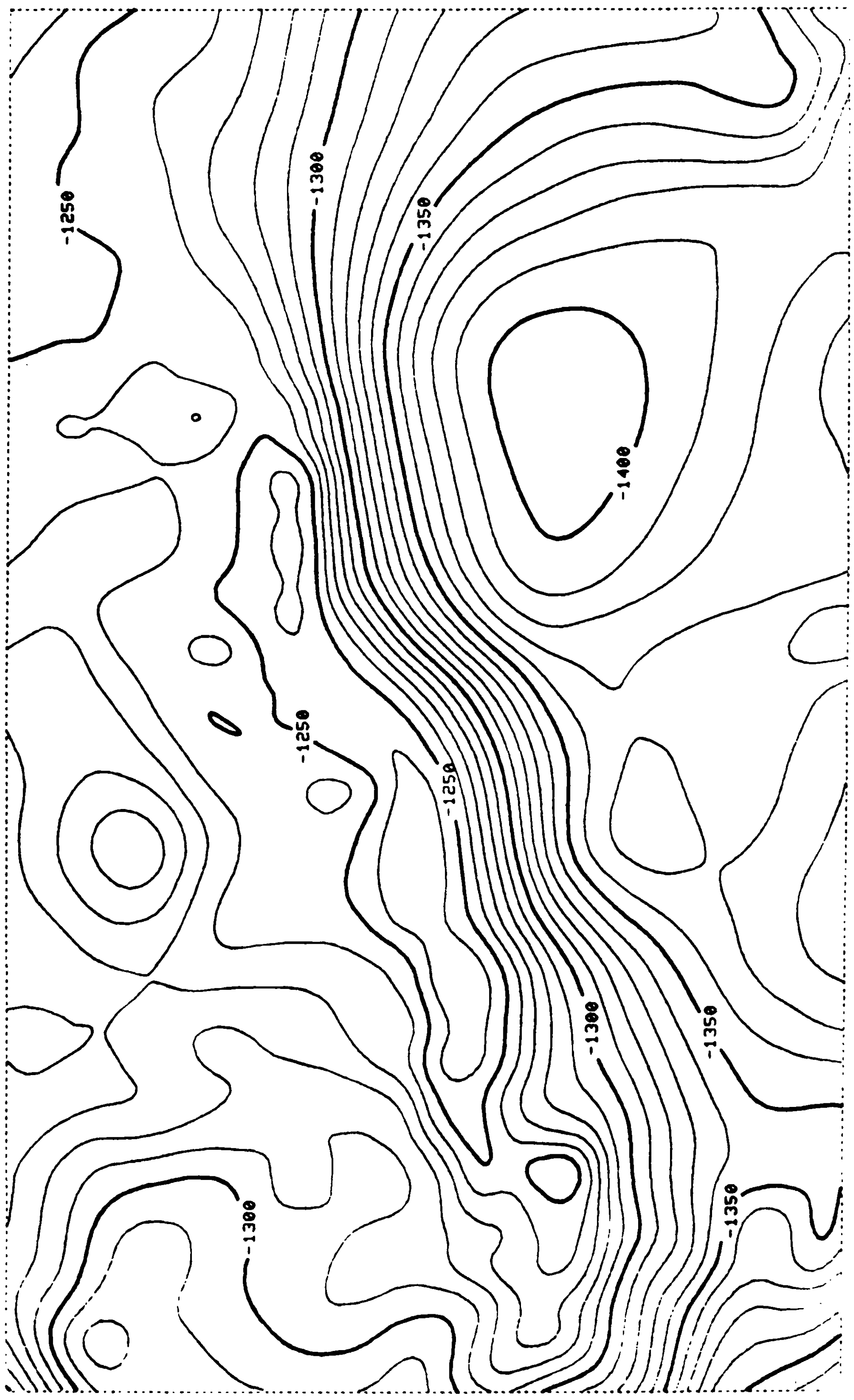


Figure 4.22 Surface II example interpolated by the $C^1$ Natural Neighbour method and contoured by CONICON (20 x 12 grid of elements).

appearance and we can therefore compare contouring methods more or less directly. Such comparisons show that the use of a 21 x 13 grid of points with the piecewise quadratic method produces contours of a much higher quality than the Surface II package, even when the latter uses the finest of grids.

# CHAPTER 5

# ERROR ANALYSIS

## 5.1 Introduction

In this chapter we study the errors involved in using the seamed quadratic element introduced in Chapter 2 to approximate well-behaved functions which it cannot reproduce exactly.

Since the errors are of order $h^3$ we concentrate on investigating that part of the error which is induced by the third order partial derivatives, and we give bounds for maximum error and integrated square error over the element in terms of these derivatives. A discussion of some of the possible applications of these bounds is then presented.

We also examine briefly the fourth order components of the error: their speedy calculation is rendered possible by the use of the CAMAL algebraic manipulation computer package (Fitch, 1982).

We begin, though, by applying Taylor's Theorem in two dimensions (see, for example, Phillips (1962) or Apostol (1969)) to find explicit forms for the third order part of the error; the forms which we derive express this error in terms of the cardinal functions of the seamed quadratic element and the four third order partial derivatives of the function which is being approximated.

### 5.1.1 Notation and Assumptions

For simplicity, we assume the element to be square in shape, with sides of length 2h parallel with the coordinate axes, and centred at the origin; results for the 2h x 2k rectangular element generalise readily from those obtained for the square element.

We define $x_i$, $i = 1,\dots, 4$ to be the vertices of the 'unit' square; that is, the square described above with h=1. Ordering of the $x_i$s is arbitrary. Each $x_i$ is a vector with components $(x_{i1}, x_{i2})$ but for convenience the conventional underlining will be dispensed with for all vectors in this section. Moreover an expression such as $x_i^2$ denotes a 2 x 2 matrix with (j, k)th component $x_{ij}\ x_{ik}$; similarly $x_i^3$ is a 2 x 2 x 2 tensor with (j, k, l)th component $x_{ij}\ x_{ik}\ x_{il}$; and so on.

The true function f which we are approximating is assumed to be at least four times continuously differentiable throughout the element. $f'$ denotes $\nabla f$, the vector of first order partial derivatives; similarly $f''$ and $f'''$ are used to denote the matrix of second order partial derivatives and the tensor of third order partial derivatives respectively. The piecewise quadratic approximant is denoted by $\hat{f}_h$ for the h-square or simply $\hat{f}$ on the unit square. The value of f at $hx_i$ is $z_i$ and the vector of partial derivatives is $s_i$. $e_h = \hat{f}_h - f$ is the error involved in using $\hat{f}_h$ to approximate f.

Finally $\lambda_i$ denotes the cardinal function corresponding to value 1 at $x_i$ and $\rho_i$ is the vector $(\rho_{i1}, \rho_{i2})^T$ of cardinal functions corresponding to partial derivatives of 1 in the x and y directions respectively at $x_i$.

### 5.1.2 Application of Taylor's Theorem

If we approximate the function f at some point x within the unit square using the seamed quadratic element we then have

$$\hat{f}(x) = \sum_{i=1}^{4} \lambda_i(x) f(x_i) + \sum_{i=1}^{4} \rho_i(x) f'(x_i) \tag{5.1}$$

In the general case, for an h-square,

$$\hat{f}_h(hx) = \sum_i \lambda_i(x) z_i + \sum_i h\rho_i(x) s_i \tag{5.2}$$

Now, taking a two-dimensional Taylor Series expansion, we have for all x within the element

$$f(hx_i) = f(hx) + h(x_i - x)f'(hx) + \frac{h^2}{2}(x_i - x)^2 f''(hx) + \frac{h^3}{6}(x_i - x)^3 f'''(hx) +$$

$$\frac{M_4}{24}\ \underline{O}\ (h^4) \tag{5.3}$$

where $M_4$ is the supremum of all five fourth order partial derivatives over the square.

Note that terms such as $(x_i - x)^3 f'''(hx)$ have an implied (triple) inner summation and are scalar quantities. Thus if $x^3$ and $y^3$ are both 2 x 2 x 2 tensors, $x^3y^3$ is defined as $\sum\sum\sum_{jkl} x^3{}_{jkl}y^3{}_{jkl}$. It follows that differentiating with respect to x gives the vector $3x^2y^3$, the lth component of which is defined as $3\sum\sum_{jk} x^2{}_{jk}y^3{}_{jkl}$, and in general terms like $x^3y^3$ can be differentiated using the conventional rules of calculus; we make use of this fact below.

Note also that the order of the final (remainder) term in (5.3) follows from inspection of the Lagrangian remainder term which in this case is $\frac{h^4}{24}(x_i-x)^4 f^{(iv)}(h\xi)$, where $\xi$ is a point on the straight line joining x to $x_i$.

Now define

$$q(x_i) = f(x) + (x_i-x)f'(x) + \tfrac{1}{2}(x_i-x)^2 f''(x) \qquad (5.4)$$

q is a quadratic function with second order contact at x with f; that is, q and its first and second derivatives are identical to f and its first and second derivatives respectively at x; generalising to the h-square we have

$$q_h(hx_i) = f(hx) + h(x_i-x)f'(hx) + \frac{h^2}{2}(x_i-x)^2 f''(hx) \qquad (5.5)$$

has second order contact with f at hx. From (5.3) and (5.5) we have

$$z_i = q_h(hx_i) + \frac{h^3}{6}(x_i-x)^3 f'''(hx) + \frac{M_4}{24}\underline{O}(h^4) \qquad (5.6)$$

Differentiating with respect to $hx_i$, we find

$$s_i = q_h'(hx_i) + \frac{h^2}{2}(x_i-x)^2 f'''(hx) + \frac{M_4}{6}\ \underline{O}\ (h^3) \qquad (5.7)$$

Now since $q_h(hx_i)$ has second order contact with f at hx it follows that the terms in $q_h$ will exactly interpolate to $f(hx) = q_h(hx)$.

$$\text{i.e.}\quad \sum_i[\lambda_i(x)q_h(hx_i) + h\rho_i(x)q'_h(hx_i)] = f(hx) \qquad (5.8)$$

Therefore, applying (5.2) to (5.6) and (5.7) we find

$$e_h(x) = \hat{f}_h(hx) - f(hx) = h^3\sum_i[{}^1/6\ \lambda_i(x_i-x)^3+\tfrac{1}{2}\rho_i(x_i-x)^2]f'''(hx) +$$

$$\frac{5M_4}{24}\ \underline{0}\ (h^4) \qquad (5.9)$$

The term in the square brackets is a 2 x 2 x 2 tensor $B_{jkl}$. We re-write (5.9) as

$$e_h(x) = h^3[c_{111}(x)\partial_{111}f(x) + 3c_{112}(x)\partial_{112}f(x) + 3c_{122}(x)\partial_{122}f(x) +$$

$$c_{222}(x)\partial_{222}f(x)] + \frac{5M_4}{24}\ \underline{0}\ (h^4) \qquad (5.10)$$

where

$$c_{111}(x) = B_{111} = \sum_i[{}^1/6\ \lambda_i(x_i-x)_1^3 + \tfrac{1}{2}\rho_{i1}(x_i-x)_1^2] \qquad (5.11)$$

$$3c_{112}(x) = B_{112} + B_{121} + B_{211} =$$

$$\sum_i[\tfrac{1}{2}\lambda_i(x_i-x)_1^2(x_i-x)_2 + \rho_{i1}(x_i-x)_1(x_i-x)_2 + \tfrac{1}{2}\rho_{i2}(x_i-x)_1^2] \qquad (5.12)$$

$$3c_{122}(x) = B_{122} + B_{212} + B_{221} =$$

$$\sum_i[\tfrac{1}{2}\lambda_i(x_i-x)_1(x_i-x)_2^2 + \tfrac{1}{2}\rho_{i1}(x_i-x)_2^2 +\rho_{i2}(x_i-x)_1(x_i-x)_2] \qquad (5.13)$$

and $c_{222}(x) = B_{222} = \sum_i[{}^1/6\ \lambda_i\ (x_i-x)_2^3 + \tfrac{1}{2}\rho_{i2}(x_i-x)_2^2]$ (5.14)

We have therefore succeeded in splitting the third order part of the error into four components, each one corresponding to one of the four third order partial derivatives. Since we know the cardinal functions $\lambda_i$ and $\rho_i$ it is possible to calculate the functions $c_{ijk}$ exactly.

## 5.2 Third order error functions

In this section we employ the CAMAL algebraic manipulation package to calculate the error functions $c_{ijk}$ derived in the previous section, and present them as piecewise cubics in x and y (we drop the vector notation of Section 5.1). We also give illustrations of the functions and discuss the significance of some of their more notable features.

Initial inspection of the expressions (5.10)-(5.13) suggests that the functions $c_{ijk}$ will be piecewise quintic in x and y, since we know that the cardinal functions $\lambda_i$ and $\rho_i$ are both piecewise quadratic in x and y. However these are the third order error terms: consequently if the function f is in fact cubic then the error surface must be a linear combination of the $c_{ijk}$, and clearly the error surface in such a case will be piecewise cubic; thus the functions $c_{ijk}$ must also be piecewise cubic, so when the expressions (5.10)-(5.13) are evaluated the fourth and fifth order terms must drop out. Calculation of these functions by hand would be a lengthy, tedious process, even if full use were made of the extensive degree of symmetry involved. Fortunately it was not necessary to carry out the evaluations manually; instead use was made of the CAMAL algebraic manipulation package (Fitch, 1982): a computer package whose capabilities include the multiplication and addition of algebraic expressions. Using this package it was necessary merely to write a single program to evaluate expressions (5.10)-(5.13) given values for the functions $\lambda_i$ and $\rho_i$, and to run the program for a minimum of two sets of values of the $\lambda_i$ and $\rho_i$.

The values of the functions $c_{ijk}$ obtained by running such a program are presented in Figures 5.1-5.4. We find that the functions $c_{ijk}$ are each composed of either two or four (and not sixteen) distinct cubic pieces, and for this reason the elements in these diagrams are pictured divided not into the sixteen constituent triangles, but just into as many pieces as there are separate cubic functions.

Note that, as we would expect, the functions $c_{122}$ and $c_{222}$ are simply reflections of $c_{112}$ and $c_{111}$ respectively in the line y = x. We therefore limit our attention to a discussion of the functions $c_{111}$ and $c_{112}$; all

Figure 5.1

$C_{111}(x)$:

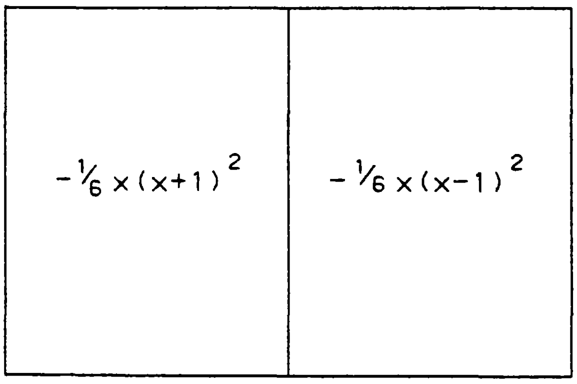


Figure 5.2

$3C_{112}(x)$:

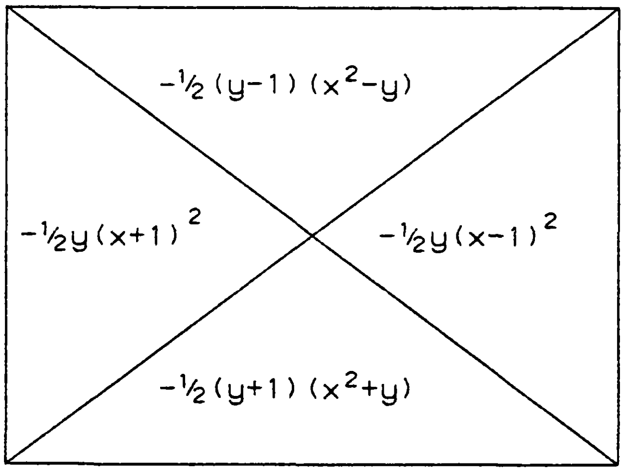


Figure 5.3

$C_{122}(x)$:

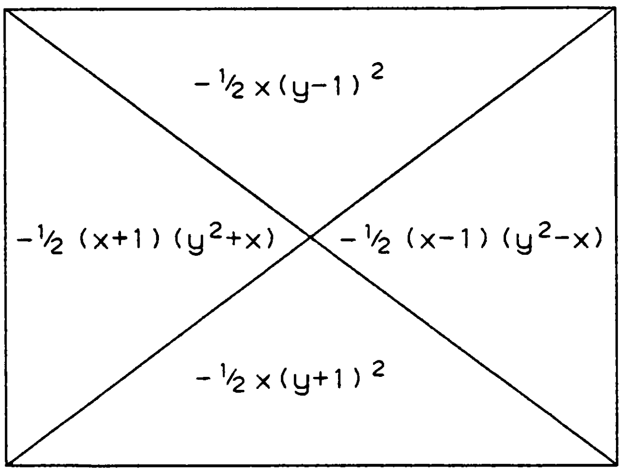
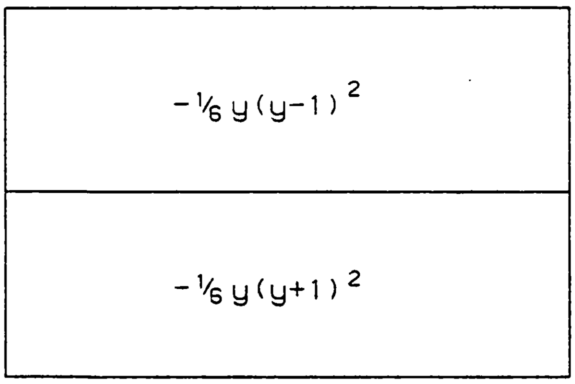


Figure 5.4

$C_{222}(x)$:



conclusions will generalise readily to the other two error functions.

Inspection of the formulae for $c_{111}$ and $3c_{112}$ (and their derivatives) reveals that both have continuous first derivatives, but have discontinuities in second derivative. In addition the third derivatives are constant everywhere except along the element's seams (where they are undefined). This is hardly surprising when we consider that the error surface which we are studying is the difference between a $C^3$ function and a $C^1$ piecewise quadratic surface with discontinuities in its second derivatives.

It can also be seen that the only third order term in the formula for $c_{111}(x, y)$ is an $x^3$ term and likewise the only third order term in the formula for $3c_{112}(x, y)$ is an $x^2y$ term. This feature is also readily explicable: consider a function f of the form $f(x, y) = \alpha x^3 + q(x, y)$, where q is a quadratic in x and y and $\alpha$ a constant. The error involved in approximating this function using the seamed quadratic element must be a constant multiple of $c_{111}(x, y)$, because there is only a single third or higher order partial derivative which is non-zero, namely $\frac{\partial^3 f}{\partial x^3}$, and this is constant throughout the domain of f. Thus $c_{111}(x, y)$ must be a constant multiple of the difference between a function of the form of f and a piecewise quadratic: hence there is only the single cubic term in $c_{111}$. A similar argument can be used to show that the only cubic term in $c_{112}(x, y)$ is an $x^2y$ term.

Pictorial representations of these functions are given in Figs. 5.5 and 5.6. For $c_{111}(x, y)$ we show a cross-sectional view for constant y rather than a contour plot because the function is a cubic in x only. Fig. 5.6 is a contour plot of $c_{112}(x, y)$.

Inspection of Figures 5.5 and 5.6 reveals other features of the functions:- $c_{111}$ has a maximum value of $^2/81$ along the line $x = {}^{-1}/3$ and a minimum of $- {}^2/81$ at $x = {}^1/3$. $3c_{112}(x, y)$ has a maximum of $^1/8$ at $(0,-\frac{1}{2})$ and a minimum of $^{-1}/8$ at $(0, \frac{1}{2})$. Both functions are zero at the origin;

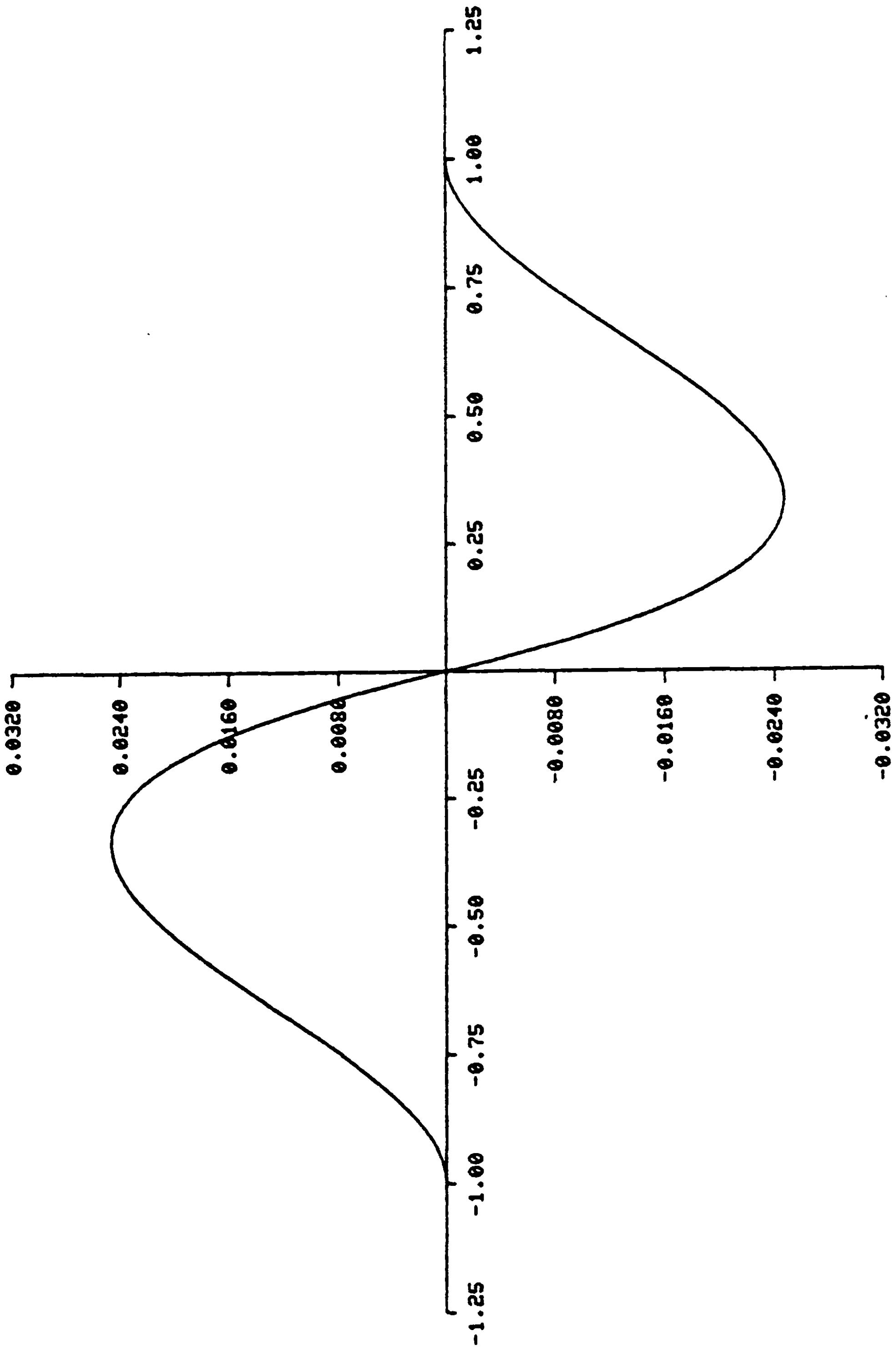


Figure 5.5 The third order error function $c_{111}$ (x, y).

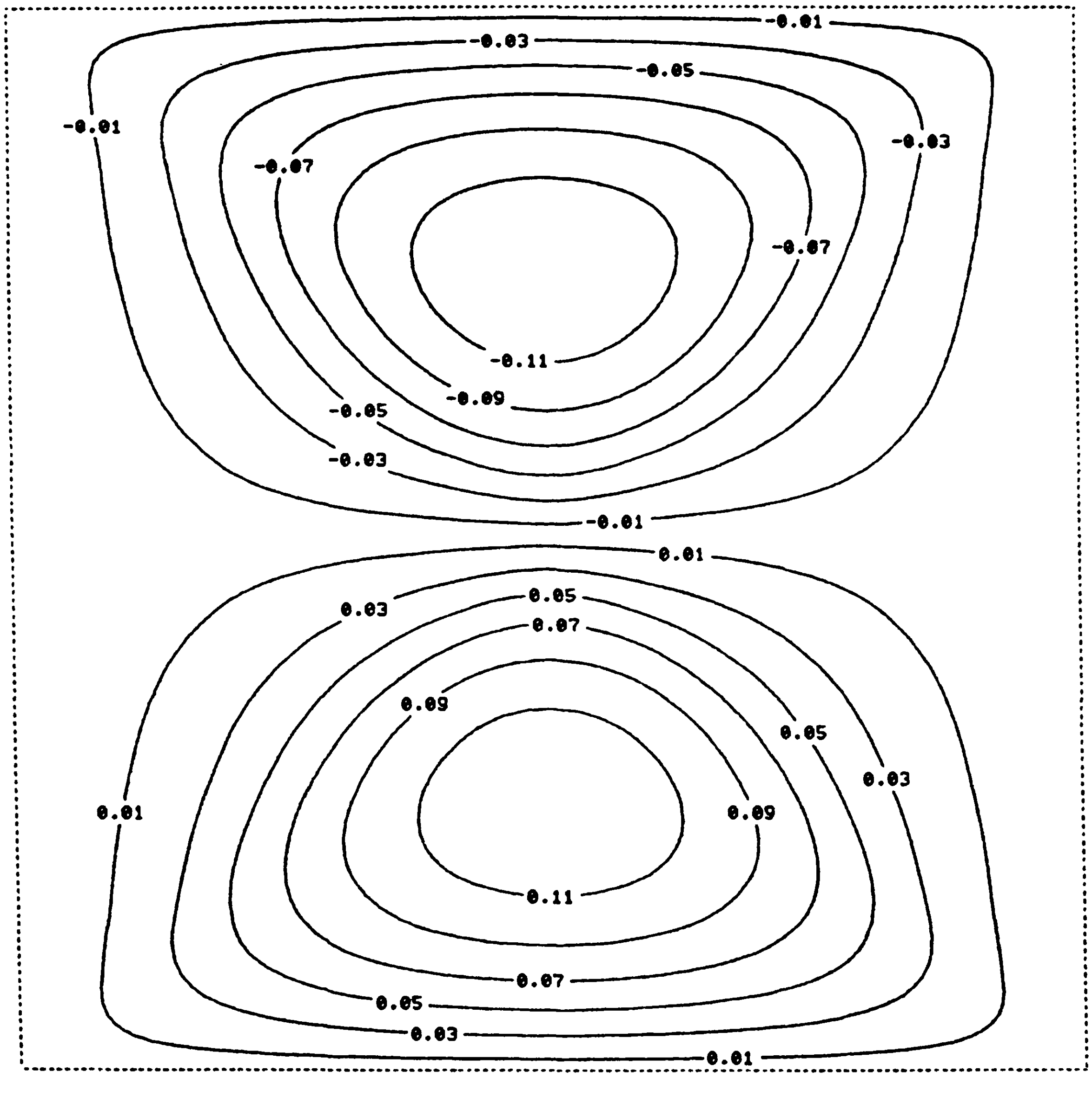


Figure 5.6 Contour plot of the third order error function $3c_{112}$ (x, y).

indeed $c_{111}(x, y)$ is zero along the y axis while $c_{112}(x, y)$ is zero along the x axis. It follows that the other two error functions will also disappear at the origin, and that the seamed quadratic element will therefore reproduce any cubic function exactly at the origin. (Note that we stated in Chapter 2 that both the seamed quadratic and seamed cubic elements have the same value at the origin; however we have a slightly stronger result here because, as a result of the gradient linearity condition, the cubic element cannot reproduce exactly a general cubic function.)

We note also that the function $c_{112}(x, y)$ disappears along the boundary of the square and has zero gradient when $x = \pm 1$. It is easy to see why this happens: consider a function f of the form $f(x, y) = \beta x^2 y + q(x, y)$ where q is quadratic in x and y and $\beta$ constant, which will lead to a constant multiple of $c_{112}(x, y)$ as error function. f is quadratic in x for constant y, and quadratic in y for constant x, so the seamed quadratic element will reproduce it exactly along its boundary. Moreover the x derivative is a linear function of y for constant x, so this too will be reproduced correctly along the lines $x = \pm 1$. We have therefore reproduced value and gradient correctly at the points (-1, 0) and (1,0), and therefore the method will also reproduce f correctly along the line joining these points; that is, the error function $c_{112}(x, y)$ is zero along the x axis.

Finally, to show why the function $c_{111}$ is zero along the y axis we recall from Chapter 2 that both one dimensional elements (seamed quadratic and cubic) yield identical values at their mid-points. Thus the one-dimensional seamed quadratic element reproduces a cubic function correctly at its mid-point. Therefore the 2-dimensional seamed quadratic element will reproduce a function of the form $f(x, y) = \alpha x^3 + q(x, y)$ correctly at the points (0, 1) and (0, -1). The y-derivative, which varies linearly in x, will also be reproduced correctly at these points. Now for fixed x, in particular $x = 0$, f is quadratic in y so once again f will be duplicated when $x = 0$; in other words the error function $c_{111}(x, y)$ will disappear along the y axis.

## 5.3 Bounds for error

The error functions calculated in Section 5.2 enable us to determine bounds for the error (maximum error or integrated square error) involved in approximating a function by the seamed quadratic element. As will be seen in the following section and in Chapter 6, a number of possible applications for such bounds suggest themselves and we therefore derive two bounds for error in this section. These are presented as Theorems 5.1 and 5.2.

We need only assume now that the function f is $C^3$ within the element. Using Lagrange's form for the remainder term and taking one term fewer in the Taylor Series expansion (5.3), we obtain

$$f(hx_i) = f(hx) + h(x_i-x)f'(hx) + \frac{h^2}{2}(x_i-x)^2 f''(hx) + \frac{h^3}{6}(x_i-x)^3 f'''(h\xi_i) \quad (5.15)$$

where $\xi_i$ is a point somewhere on the straight line joining x to $x_i$. Expression (5.9) then becomes

$$e_h(x) = h^3 \sum_i [1/6\, \lambda_i (x_i-x)^3 + \tfrac{1}{2}\rho_i (x_i-x)^2] f'''(h\xi_i) \quad (5.16)$$

Thus the total error has been expressed in terms of the third order partial derivatives at four points within the square. We can therefore derive expressions giving bounds for maximum error and integrated square error over the element in terms of the supremum of the moduli of the four third order partial derivatives, which we shall refer to as $M_3$.

### 5.3.1 Maximum error

We note that the functions $c_{111}(x, y)$ and $c_{122}(x, y)$ are symmetric about the x-axis and antisymmetric about the y-axis, and both are positive when $x < 0$ and negative when $x > 0$. The functions $c_{112}(x, y)$ and $c_{222}(x, y)$, being reflections in the line $y = x$ of $c_{122}(x, y)$ and $c_{111}(x, y)$ respectively, have similar properties but with directions reversed. It follows that calculation of a bound for maximum error in terms of $M_3$ is a trivial operation:- the worst case will clearly occur when the partial derivatives are equal in modulus, and since the four error functions are all positive throughout the part of the element which lies in the third quadrant we simply need to maximise the sum of the error functions (that is $c_{111}(x, y) + 3c_{112}(x, y) + 3c_{122}(x, y) + c_{222}(x, y)$) in this region. In fact the maximum occurs at $(-{}^1/_3, -{}^1/_3)$ and has a value of ${}^{16}/_{81}$. We therefore have the bound for maximum error within the element given by the following theorem.

<u>Theorem 5.1</u>

If a seamed quadratic element of dimension 2h x 2h, centred at the point (a, b), is used to approximate a $C^3$ function f, then the maximum error in approximation is bounded by the following:

$$\text{maximum error} \leq {}^{16}/_{81}\, h^3 M_3 \qquad (5.17)$$

$$\text{where } M_3 = \sup_{\substack{|x-a| \leq h \\ |y-b| \leq h}} \left( \left|\frac{\partial^3 f}{\partial x^3}\right|, \left|\frac{\partial^3 f}{\partial x^2 \partial y}\right|, \left|\frac{\partial^3 f}{\partial x \partial y^2}\right|, \left|\frac{\partial^3 f}{\partial y^3}\right| \right)$$

Note that, ignoring the trivial case where f is quadratic, this bound can only be attained at the points $(\pm {}^1/_3, \pm {}^1/_3)$, and if it is to be attained we require that the greatest magnitude of each of the third order partial derivatives should occur at all of the points $\xi_i$ corresponding to this particular value of x; at these points the third order partial derivatives must be equal in modulus with sign appropriate to the particular quadrant in which x lies. The bound is clearly attainable: for example it would be attained at both $({}^1/_3, -{}^1/_3)$ and $(-{}^1/_3, {}^1/_3)$ if f were of the form

$$f(x, y) = \alpha({}^1/_6 x^3 - \tfrac{1}{2}x^2 y + \tfrac{1}{2}xy^2 - {}^1/_6 y^3) + q\,(x, y)$$

where $\alpha$ is a constant, and q is a quadratic function in x and y.

5.3.2 <u>Integrated square error</u>

As a consequence of (5.16), the following expression will clearly be a bound for the integrated square error (i.s.e.) over the element:-

$$\text{i.s.e.} \leq M_3^2 h^8 \int_{-1}^{1}\int_{-1}^{1}(|c_{111}(x, y)| + 3|c_{112}(x, y)| + 3|c_{122}(x, y)| + |c_{222}(x, y)|)^2 dx\, dy$$

The component parts of this integral are as follows:

$$\int_{-1}^{1}\int_{-1}^{1} c_{111}^2 (x, y)\, dx\, dy = \int_{-1}^{1}\int_{-1}^{1} c_{222}^2(x, y) dx\, xy = \frac{1}{945}$$

$$9 \int_{-1}^{1}\int_{-1}^{1} c_{112}^2 (x, y)\, dx\, dy = 9 \int_{-1}^{1}\int_{-1}^{1} c_{122}^2(x, y) dx\, dy = \frac{4}{315}$$

$$6 \int_{-1}^{1}\int_{-1}^{1} |c_{112}(x, y) c_{111}(x, y)| dx\, dy = 6 \int_{-1}^{1}\int_{-1}^{1} |c_{122}(x, y) c_{222}(x, y)| dx\, dy = \frac{11}{1890}$$

$$6 \int_{-1}^{1}\int_{-1}^{1} c_{111}(x, y) c_{122}(x, y) dx\, dy = 6 \int_{-1}^{1}\int_{-1}^{1} c_{112}(x, y)\, c_{222}(x, y) dx\, dy = \frac{1}{189}$$

$$2 \int_{-1}^{1}\int_{-1}^{1} |c_{111}(x, y) c_{222}(x, y)| dx\, dy = \frac{1}{648}$$

$$18 \int_{-1}^{1}\int_{-1}^{1} |c_{112}(x, y) c_{122}(x, y)| dx\, dy = \frac{3}{280}$$

It follows that

$$\int_{-1}^{1}\int_{-1}^{1}(|c_{111}(x, y)| + 3|c_{112}(x, y)| + 3|c_{122}(x, y)| + |c_{222}(x, y)|)^2\, dx\, dy$$

$$= \frac{703}{11340} \simeq 0.062$$

and therefore we have the bound for i.s.e. given in Theorem 5.2.

Theorem 5.2

If a seamed quadratic element of dimension 2h x 2h, centred at the point (a, b), is used to approximate a $C^3$ function f, then the integrated square error in approximation is bounded by the following:

$$\text{i.s.e.} \leq \frac{703}{11340} h^8 M_3^2 \qquad (5.18)$$

($M_3$ defined as in Theorem 5.1).

Unlike the bound derived for maximum error, this bound is not attainable except in the trivial case where f is quadratic. For if it were attainable we would require f to be a piecewise cubic of the form

$$f(x, y) = \begin{cases} \alpha(-{}^1/6x^3 - \frac{1}{2}x^2y - \frac{1}{2}xy^2 - {}^1/6y^3) + q_1(x, y) & x \geq 0, y \geq 0 \\ \alpha({}^1/6x^3 - \frac{1}{2}x^2y + \frac{1}{2}xy^2 - {}^1/6y^3) + q_2(x, y) & x \leq 0, y \geq 0 \\ \alpha({}^1/6x^3 + \frac{1}{2}x^2y + \frac{1}{2}xy^2 + {}^1/6y^3) + q_3(x, y) & x \leq 0, y \leq 0 \\ \alpha(-{}^1/6x^3 + \frac{1}{2}x^2y - \frac{1}{2}xy^2 + {}^1/6y^3) + q_4(x, y) & x \geq 0, y \leq 0 \end{cases}$$

where $q_i$, i = 1,..., 4 are quadratic in x and y and $\alpha$ is a constant. However such a function, though it could be made continuous, could not be made $C^2$ or even $C^1$ for non-zero $\alpha$. Therefore this bound is only attained when f is reproduced perfectly.

Thus, to summarize this section, we have found a pair of bounds, one for maximum error and the other for integrated square error over the element. In the non-trivial case where the error is non-zero the former is attainable, but the latter is not.

## 5.4 Discussion

The bounds for error determined in the previous section may be put to practical effect in a number of different ways, and in this section we discuss some of the possibilities.

We begin by considering how the bounds may be used in the analysis of contour plots, to determine the error involved in contouring a function using the piecewise quadratic approximant; or in their design, to provide an automatic method of choosing the grid size. In addition we explain how the bounds may be employed to provide a criterion for local splitting of grid squares so that the grid size is allowed to vary over different areas of a single plot, according to the behaviour of the function.

In the second part of the discussion we consider combining the bounds with some measure of slope over an element to provide a criterion for local splitting which depends not simply on vertical error in approximation but on 'horizontal error', or vertical error relative to local slope. We examine a number of possible measures for slope in an element and arrive at a recommendation for the use of one of these. We then derive a possible index for horizontal error within an element from a combination of this measure and our bounds for error.

Finally we discuss the relative merits of using vertical error or horizontal error as criteria for local splitting of grid elements. Mainly as a result of practical considerations, we prefer the use of the former.

### 5.4.1 Using bounds for error alone

Probably the most obvious application of the bounds is in the analysis of contour maps produced by the seamed quadratic element: if it is possible to determine the maximum (in modulus) of each of the function's third order partial derivatives in the area of the plot, then a bound for the maximum error involved in the contouring process can immediately be found using the bound given by Theorem 5.1. To obtain a useful bound for integrated square error, however, we require the maxima of the third order partial derivatives in each element used in the plot. Calculation of these maxima may be a formidable task, and this is one reason why we shall concentrate most of our attention on maximum error rather than integrated square error. A further reason for doing so is that non-zero bounds for maximum error, unlike those for integrated square error, are attainable in at least a few cases and are therefore perhaps likely to be closer to the true error in general. But probably the most important reason is that the non mathematically-minded scientist using the seamed quadratic element to produce contour plots is likely to find the concept of maximum error much simpler to grasp intuitively than that of integrated square error, and he will therefore be less likely to misunderstand the information given to him by these bounds.

We return now to Figure 5.6, which is a contour plot of the function $3c_{112}(x,y)$, a piecewise cubic in x and y. The error in this example is the same as that for the cubic $-\frac{1}{2}x^2y$, since the piecewise quadratic part of the function is reproduced correctly given our 8 x 8 grid of elements. Thus $M_3 = 1$, and expression 5.17 provides a bound for maximum error over the whole plot of $^{16}/_{81} \cdot (^1/_8)^3 \cdot 1 \approx 0.00039$. In this particular example it is also simple to compute the bound for integrated square error using expression 5.18. This will be

$$\frac{703}{11340} \cdot 1 \cdot (^1/_8)^8 \cdot 64 \approx 2.365e\text{-}07$$

As a result of the special nature of the example chosen here, it is a simple task to evaluate exactly for comparison with our bounds both the maximum error and the integrated square error involved in approximating the function by the piecewise quadratic element. The maximum error is given by

$^1/_8 \cdot (^1/_8)^3 \cdot 1 \approx 0.00024$ (since $^1/_8$ is the maximum value of $3c_{112}(x, y)$) and the integrated square error by

$$64\, M_3^2\, h^6\, 9\int_{-1}^{1}\int_{-1}^{1} c_{112}^{\,2}(x, y)\, dx\, dy = 64 . 1 . (^1/_8)^8 . \frac{4}{315} \approx 4.844e\text{-}08$$

However in the vast majority of cases it will not be possible to calculate the error exactly and the bounds derived in the previous section will have to suffice. Note though that in the example above the bound for maximum error gives a better approximation (in relative terms) to the true maximum error than the bound for integrated square error gives to the true i.s.e.

As an alternative to using the bounds as a tool in the analysis of a contour plot they may also be used in the design stages of the contouring process. Thus, if we know the maximum third order partial derivative of the function to be contoured in the area of interest, then we can use the bound for maximum error to determine the largest grid size which will result in the approximant never differing from the true function by more than a fixed amount, say $\delta$, which in theory can be set as small as we wish (in practice limitations will be set by the availability of memory and CPU time). Thus, if we wished to contour the function $3c_{112}(x, y)$ with maximum error no greater than 0.0002 then we would require

$$\frac{16}{81} h^3 . 1 \leq 0.0002$$

i.e. $h \leq 0.1004$

Therefore, since unity must be an integer multiple of h, we would choose $h = 0.1$ and use a 10 x 10 grid of elements for our plot. Note that, as a result of the third order nature of the error, a near 50% reduction in the size of the bound results from a 20% reduction in grid size.

Using the bound for maximum error in the design of the contour map thus provides an automatic method of choosing the grid size, which solves one of the major problems faced by anyone wishing to produce a contour map of a known function using the CONICON software.

In an example such as the one which we have chosen we could similarly control integrated square error; note that the reduction in integrated square error is proportional to $h^6$ rather than $h^8$ since the number of elements in the plot increases proportionally to $1/h^2$.

Of course in the vast majority of practical applications it will not be possible to determine the maxima of the third order partial derivatives analytically; instead numerical methods will have to be employed. In some cases it may prove too difficult or too costly even to apply numerical methods. In such examples the only alternative is to approximate the maxima in some way. Since the bound for maximum error is not usually exact, a good approximation of maximum third order partial derivative will in the vast majority of cases result in bounds which do contain the true maximum error. If the bound is to be utilised in the analysis rather than the design of a contour map, then the maximum (in modulus) of the third order partial derivatives at all grid points will usually give a good enough approximation. However when it is wrong it will underestimate the true maximum and as a result of this the (remote) possibility that the bound does not contain the true maximum error cannot be ruled out; and the certainty is thus removed from our conclusions. If the bound is to be employed in the design stage of the contour plot then uncertainty about its validity is probably less important: in such circumstances the user will usually be more concerned with making a sensible choice of grid size than with precise knowledge of the maximum error involved in the approximation.

If it is possible to find (either exactly or approximately) the maximum of the third order partial derivatives within any element of the contour map, then either of the bounds can be used to form a criterion for local splitting of grid squares, rather than being used to determine the overall (constant) grid size: so long as the opportunity to make additional function and gradient evaluations exists, it is quite possible to split any existing elements of the grid into four, retaining continuous differentiability of the surface, and to continue to do so until all elements in the grid, whatever their size, fulfil a certain criterion. A suitable criterion could obviously be that the maximum error (or i.s.e.

per unit area) within each element is less than a certain value. Rather than splitting all elements in the grid until this is achieved it is preferable for reasons of efficiency to split only those elements which do not fulfil the criterion and to carry on doing so recursively until all elements satisfy it.

### 5.4.2 Combining with measures of slope

Until this point we have considered only the vertical error involved in approximating the true function by the seamed quadratic element. However it can be argued that this is not what we should be examining: if we require our contours to be close in some sense to the true contours then it is the horizontal error which we must keep under control. 'Horizontal error' is a rather difficult concept to define (for example, in some cases a contour which exists in the approximant may have no contour to correspond to it in the true surface), but it can be thought of, loosely speaking, as vertical error relative to the local slope, since a vertical error of $\delta$ in an area where the surface slopes steeply will result in a much smaller error in the contours than will a vertical error of $\delta$ in an area where the surface is nearly flat.

Thus we would like to have some sort of measure to summarise the slope within an element, to accompany the bound for maximum error within that element; in particular we are concerned with the gradient in parts of the element where the surface is relatively flat, so our measure ought to be a measure (even if only a crude one) of the flattest triangle within an element. Note that a triangle in which a maximum or minimum occurs will not necessarily be considered to be as flat as one in which the surface slopes very gently throughout. For simplicity it would be convenient if the slope within an element could be summarised by a single simple function of the data at the vertices. Averaging the sums of squares of the two partial derivatives at each vertex will not result in a reliable measure, for each of the cardinal functions is zero in the two triangles opposite the vertex at which the non-zero datum occurs; thus if the data are zero at three vertices of the element then, whatever the data at the fourth vertex are, the surface will be flat over at least two of the element's constituent triangles. However the measure suggested would give no indication of this fact if either of the derivatives at the fourth vertex were large.

It might appear that this problem could be solved by taking the average of the three (or two) smallest sums of squares of partial derivatives at the vertices. However flat areas might still occur undetected by such a measure: Figure 5.7 shows an example in which the surface is flat over a single triangle of the element, and the partial derivatives are both zero at a single vertex of the element. The unlabelled contours in this diagram are at $1.0 \pm 0.0001$. In fact it is possible for flat areas to occur when all the gradients at the vertices are non-zero, and such an example is illustrated in Figure 5.8. The flat areas in this map will occur given any constant multiple of the data at the vertices, so clearly a different measure of flatness will have to be considered. To derive a suitable measure it is helpful to examine the conditions required on the data which will result in the approximant being constant over a triangle. Using the cardinal functions we can evaluate the coefficient of each term of the quadratic in a particular triangle in terms of the data at the vertices. For the function to be a constant we require that the coefficients of $x^2$, $y^2$, $xy$, $x$ and $y$ are all zero; thus we have five linear conditions on the data. Figure 5.9 numbers two of the triangles within an element 1 and 2, and labels the vertices of the subsquares which form the element in an obvious way.

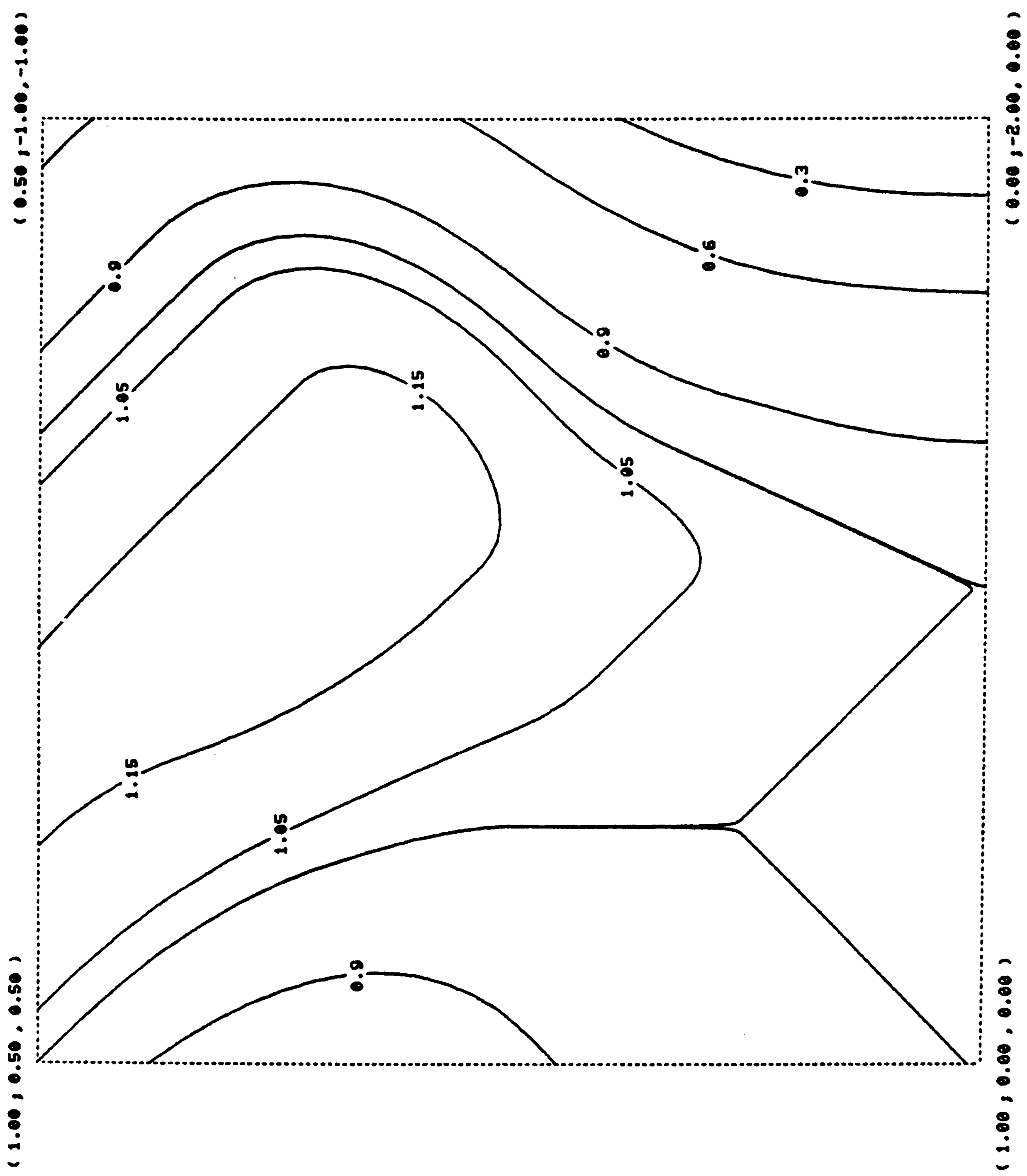


Figure 5.7 A single element illustrating constant approximant value across an 'external' triangular panel.

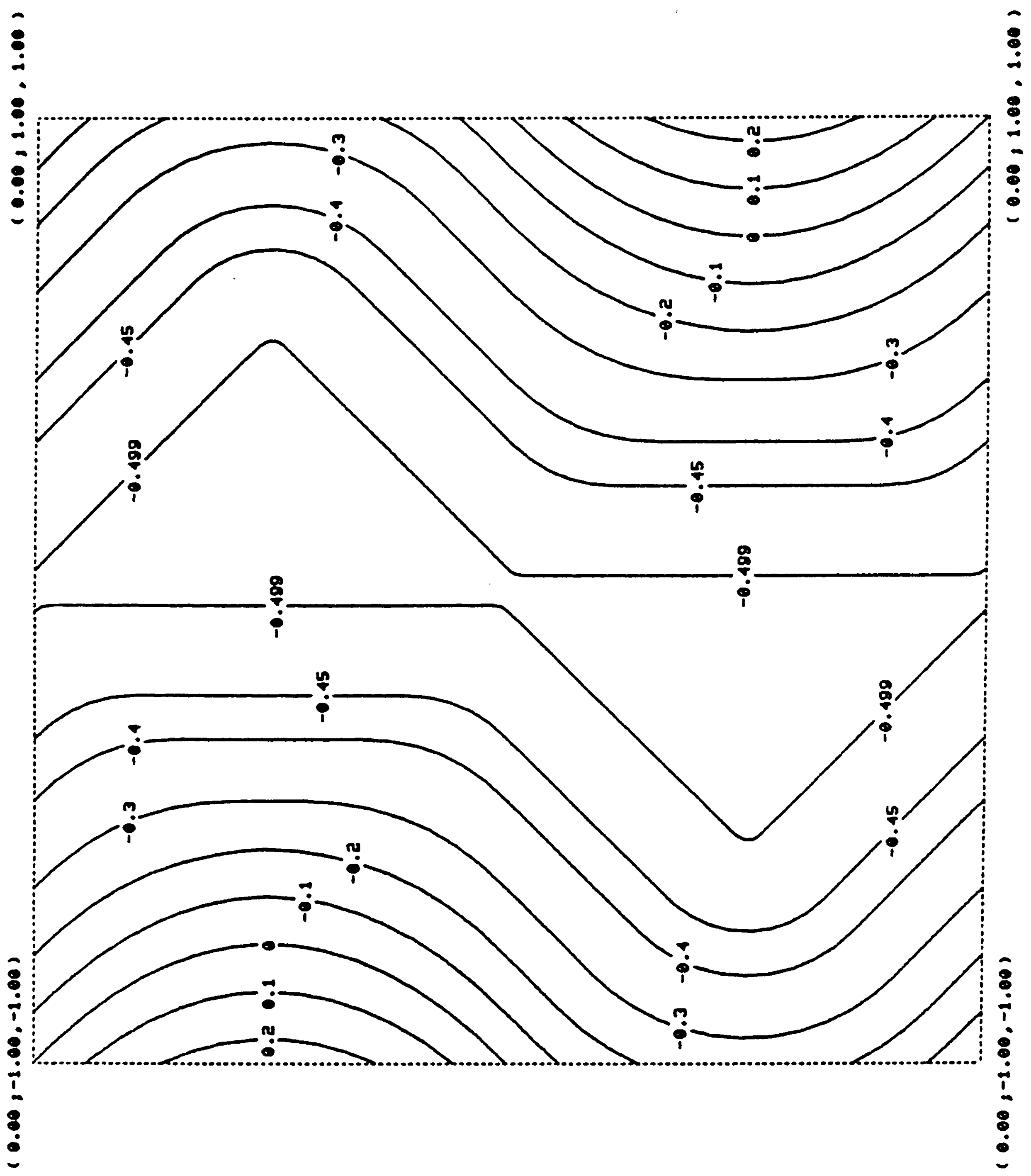
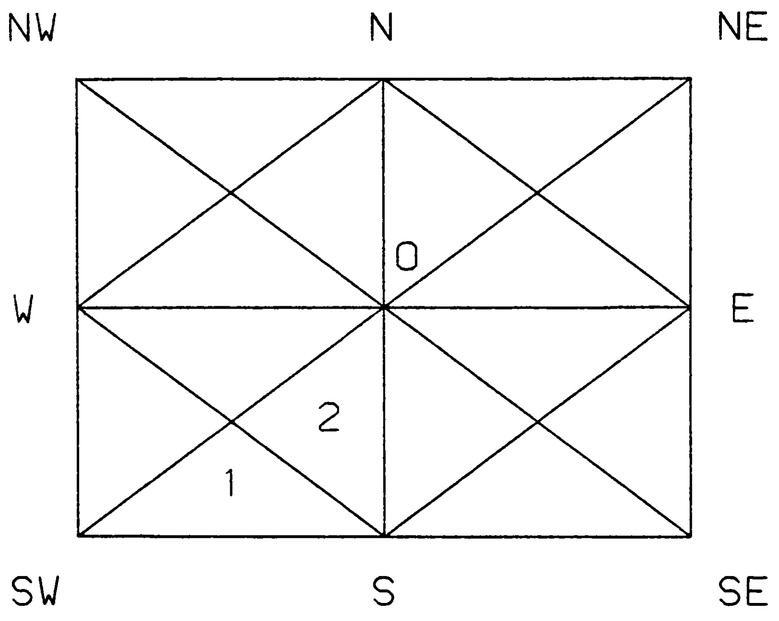


Figure 5.8 A single element illustrating constant approximant value over two 'internal' triangular panels.

Figure 5.9



We define $s_{SW}$ and $t_{SW}$ to be the partial derivatives in the SW corner of the element in the x- and y-directions respectively, and gradients at all other vertices of the element are given a similar notation. The gradients at N, E, S, W and O can all be calculated as linear combinations of the data at the vertices and we refer to these as $s_N$, $t_N$, $s_E$ and so on. Now by examining the coefficients of $x^2$, $y^2$, etc. in the quadratic on triangle 1, and then taking linear combinations of the resulting conditions, we find the following five conditions for the quadratic on triangle 1 to be constant:-

(i) $s_{SW} = 0$ (ii) $t_{SW} = 0$

(iii) $s_S = 0$ (iv) $t_S = 0$

(v) $s_W = -t_W$ (5.19)

(Note that the data in Figure 5.7 can be shown to satisfy these conditions.)

The reasons for conditions (i) to (iv) are simple to grasp intuitively but no obvious intuitive reason offers itself for the final condition:- it should be noted though that the negative coefficient of $t_W$ means that the two partial derivatives at W in the direction of triangle 1 are equal.

For triangle 2 we have a similar set of conditions:

(i) $s_S = 0$ (ii) $t_S = 0$ (iii) $s_0 = 0$

(iv) $t_0 = 0$ (v) $s_{SW} = t_{SW}$ (5.20)

Note that in this case none of the gradients at the vertices are required to be zero; the data in Figure 5.8 can be shown to satisfy these conditions.

All triangles within the element are essentially of the same type as triangle 1 (one side forms part of the element's border) or triangle 2 (one vertex touches 0) and the conditions for the surface to be flat over any triangle will be similar to those given for triangles 1 and 2. Therefore, in order to be certain that the surface is nowhere constant within an element it will normally suffice to check gradients at the vertices and centre of the element; occasionally we might also have to check gradients at N, E, S or W. (For the remainder of this discussion, the 'gradient' at a point will denote the sum of squares of the two first order partial derivatives at that point.) This suggests that a crude measure of flatness might be obtained from some function of the gradients at the vertices and centre of the element.

The simplest such measure to suggest itself would appear to be the minimum of these five gradients. Such a measure could be computed very quickly and would automatically be zero if the surface was constant over any of the sixteen triangles of the element. However, as a result of its simplicity, it could give misleading values in a number of cases: for example, if a local maximum or minimum of the surface were to occur close to one of the points at which gradients were being evaluated.

The following algorithm will produce an alternative measure which should always give reliable results:

(1) Evaluate gradients at the vertices of all sixteen triangles (thirteen points in all).

(2) Find the maximum of the gradients at the vertices of each triangle.

(3) Regard the infimum of these sixteen values as the measure of flatness.

This measure might however prove to be too much of a burden in terms of the amount of computation required, and so a third measure of flatness has been selected as a compromise between the two already mentioned. This is evaluated in the following way:-

(1) Calculate gradients at the nine subelement vertices.

(2) Find the pair of neighbouring vertices with the smallest average gradient.

(3) Select this average gradient as a measure of flatness.

Such a measure will, as a result of conditions (5.19) and (5.20), always equal zero if the surface is constant over any triangle, but will not usually be close to zero if a local maximum or minimum happens to occur in the neighbourhood of a vertex or the centre of an element. The measure is guaranteed to be small in cases where a flat triangle exists and will usually be large if a flat triangle does not exist; thus any errors which do occur will lead only to elements being divided unnecessarily, on outcome which is preferable to the alternative of neglecting to divide elements which ought to be split up.

The reader will recall that our measure of flatness was chosen in order to be used in combination with the bound for maximum error in forming a criterion for local splitting of elements, and therefore an expression which combines the two has to be selected. We require something similar to the quotient of maximum error over (flatness of) gradient, but this ratio cannot be used as it stands because our gradient measure may sometimes turn out to be zero. Note also that in cases where the surface is constant along the edge of an element then, irrespective of the number of times that the element and its constituent parts are subdivided, the measure will always equal zero for some elements. Thus to guarantee termination of the process as well as to avoid division by zero it seems appropriate to add a positive constant parameter $\kappa$ to the denominator of the quotient. The measure of error involved in approximating f by the piecewise quadratic element then becomes:

$$\frac{\text{bound for maximum error}}{\text{measure of flatness} + \kappa}, \qquad \kappa > 0$$

and we will decide to subdivide an element into four if this measure turns out to be greater than a certain positive constant, $\varepsilon$. $\kappa$ will be chosen in such a way that when the measure of flatness is zero, the process will terminate at or before a selected grid size. Thus if we do not wish to split elements more than j times we will choose $\kappa$ in such a way that

$$\frac{16}{81} \cdot \frac{h_j^{\,3} M_3}{\kappa} < \varepsilon$$

but

$$\frac{16}{81} \cdot \frac{h_{j-1}^{\,3} M_3}{\kappa} > \varepsilon$$

where $h_j$ is the size of an element which has resulted from j splitting processes ($h_j = \frac{1}{2}h_{j-1}$), and $M_3$ refers to the maximum third order partial derivative for the entire area of interest.

5.4.3 Horizontal vs Vertical error

Unfortunately the practicability of using a criterion such as the one derived above for local splitting of grid squares is very much open to question. Probably its greatest disadvantage is that the user is faced with the difficult task of selecting the values of two parameters, while if only the vertical error was considered a single parameter would have sufficed. In addition, it may well be the case that the approximations involved in measuring maximum error and flatness in an element will, when combined, result in poor behaviour of the method. It is perhaps therefore worth discussing the relative importance of horizontal and vertical error in contour plots at this point.

The case for treating horizontal error as more important has already been advanced in this discussion: it is that large discrepancies between the contours of the approximant and those of the true function may pass unnoticed in flat areas if no account is taken of local slope. However vertical error has the advantage that it can be defined precisely, which horizontal error cannot be, and we have shown that it is the easier error to measure and can be measured reasonably accurately. If we know that the surface being contoured is never farther than a small amount $\delta$ vertically from the true surface then perhaps there is little cause for concern when contours depart significantly from their true positions; in areas where this happens the contours will in any case be large distances apart and the experienced reader of contour plots will realise the dangers of inferring very much from their positions. Such contours, although unreliable in terms of distance from the true contours, will still give an accurate representation of the nature of the surface in these areas.

Besides the practical obstacles towards implementing a method which chooses the grid size on the basis of a measure of horizontal error, a number of other problems exist which might lead us to doubt the wisdom of proceeding with such a method. Firstly, any measure of horizontal error, though it may be suited to local refinement of grid size, is likely to be extremely difficult to use to select an overall grid size (since many surfaces which would be well approximated by a large grid for the most part include areas where the function is very flat); secondly, it is much more difficult to gain an intuitive appreciation of measures of horizontal error than is possible in the case of vertical error; and finally, a method which tended to split up elements more often in areas where the surface is relatively flat would probably be wasteful because, as a direct consequence of the flatness of these areas, very few contours would pass through them (assuming contour intervals to be constant).

It is therefore felt, for the various reasons outlined above, that the extra effort required to measure horizontal error in contours will in most cases probably not be justified by improved results. However, in the following Chapter, we shall attempt to put to practical use some of the measures of both horizontal and vertical error derived in this section.

## 5.5 Fourth order error functions

In this section we present the fourth order equivalents of the error functions given in Section 5.2. These error functions are not as useful as the third order error functions in that we are unlikely to wish to use them to construct bounds for error in the way that we used the third order error functions in Section 5.3; however, by comparing these functions with each other it is possible to gain some insight into the relative importance of the various fourth order partial derivatives in determining the error at any given point in the element.

In order to derive these functions we must assume that f is five times continuously differentiable throughout the element. We take an extra term in the Taylor Series expansion (5.3), which results in (5.9) becoming the following:

$$e_h(x) = h^3 \sum_i [1/6\ \lambda_i (x_i-x)^3 + \tfrac{1}{2}\rho_i (x_i-x)^2] f'''(hx) + h^4 \sum_i [1/24\ \lambda_i (x_i-x)^4 +$$

$$1/6\ \rho_i (x_i-x)^3] f^{(iv)}(hx) + \frac{M_5}{20} \underline{0}\ (h^5) \tag{5.21}$$

$M_5$ here is defined in an analogous manner to $M_3$ and $M_4$; and with the obvious extension of notation, the fourth order equivalents of expressions (5.11)-(5.14) are thus as follows:-

$$c_{1111}(x) = \sum_i \{1/24\ \lambda_i (x_i-x)_1^4 + 1/6\ \rho_{i1} (x_i-x)_1^3\} \tag{5.22}$$

$$4c_{1112}(x) = \sum_i \{1/6\ \lambda_i (x_i-x)_1^3 (x_i-x)_2 + \tfrac{1}{2}\rho_{i1} (x_i-x)_1^2 (x_i-x)_2 + 1/6\ \rho_{i2} (x_i-x)_1^3\} \tag{5.23}$$

$$6c_{1122}(x) = \sum_i \{\tfrac{1}{4}\lambda_i(x_i-x)_1^2(x_i-x)_2^2 + \tfrac{1}{2}\rho_{i1}(x_i-x)_1(x_i-x)_2^2 + \tfrac{1}{2}\rho_{i2}(x_i-x)_1^2(x_i-x)_2\} \tag{5.24}$$

$$4c_{1222}(x) = \sum_i \{ 1/6\, \lambda_i(x_i-x)_1(x_i-x)_2^3 + 1/6\, \rho_{i1}(x_i-x)_2^3 + \tfrac{1}{2}\rho_{i2}(x_i-x)_1(x_i-x)_2^2\} \tag{5.25}$$

$$c_{2222}(x) = \sum_i \{ 1/24\ \lambda_i(x_i-x)_2^4 + 1/6\ \rho_{i2}(x_i-x)_2^3\} \tag{5.26}$$

Once more the expressions appear to be of a higher order than expected (sextic rather than quartic), but the fifth and sixth order terms all drop out when functions (5.22) to (5.26) are calculated. Use was again made of the CAMAL algebraic manipulation package in evaluating these functions, with an even greater saving in time and effort than resulted from its earlier use; indeed it would probably not have been a practicable proposition to find the functions $c_{jklm}$ without the use of such a package in the context of a project of this nature.

The functions $c_{1111}(x, y)$, $6c_{1122}(x, y)$ and $4c_{1112}(x, y)$ are presented in Figures 5.10-5.12 respectively (returning once again to conventional Cartesian notation); the functions $4c_{1222}(x, y)$ and $c_{2222}(x, y)$ are omitted because they are simply reflections of $4c_{1112}(x, y)$ and $c_{1111}(x, y)$ respectively in the line $y = x$. $c_{1112}$ is the most complex error function encountered in this chapter in the sense that it is the only one to be composed of sixteen distinct polynomials; for this reason we employ a key numbering the triangles which comprise the element from 1 to 16.

Inspection of the formulae for $c_{1111}$, $4c_{1112}$ and $6c_{1122}$ (and their derivatives) reveals that they are each $C^2$ with discontinuities in third derivative. The functions display similar behaviour to the third order error functions in that $x^4$ is the only fourth order term in the formula for $c_{1111}(x, y)$, $x^3y$ is the only fourth order expression in $c_{1112}(x, y)$, and so on. This can be explained using an argument only slightly more

Figure 5.10

$c_{1111}(x,y)$:

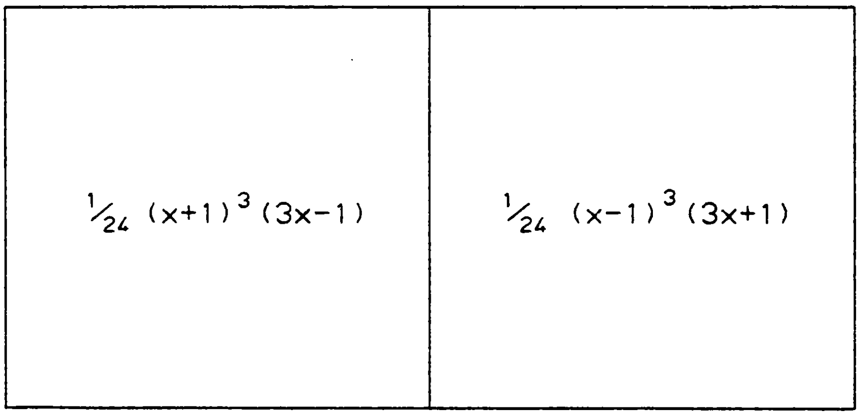
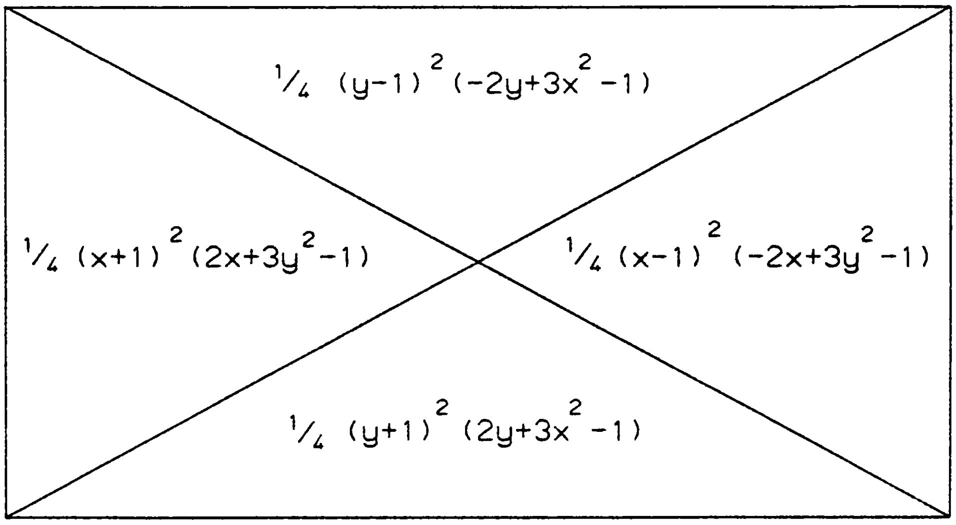


Figure 5.11

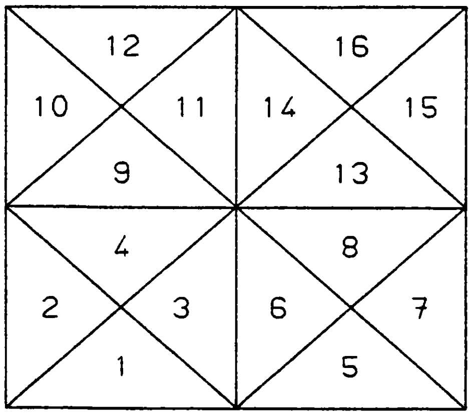
$6c_{1122}(x,y)$:



$\frac{1}{4}(y-1)^2(-2y+3x^2-1)$
$\frac{1}{4}(x+1)^2(2x+3y^2-1)$
$\frac{1}{4}(x-1)^2(-2x+3y^2-1)$
$\frac{1}{4}(y+1)^2(2y+3x^2-1)$

Figure 5.12

Key:



| TRIANGLE NO. | VALUE OF FUNCTION $4c_{1112}(x,y)$ |
|---|---|
| 1 | $\frac{1}{6}(y+1)(3x+1)(x^2+y) + \frac{1}{6}(y+1)(x+1)^2$ |
| 2 | $\frac{1}{6}(x+1)^2(3xy+2y+1)$ |
| 3 | $\frac{1}{6}x(3(y+1)(x^2+y) + x(2y+1))$ |
| 4 | $\frac{1}{6}y(-y + x(3x+5)(x+1))$ |
| 5 | $\frac{1}{6}(y+1)(3x-1)(x^2+y) - \frac{1}{6}(y+1)(x-1)^2$ |
| 6 | $\frac{1}{6}x(3(y+1)(x^2+y) + x(-2y-1))$ |
| 7 | $\frac{1}{6}(x-1)^2(3xy-2y-1)$ |
| 8 | $\frac{1}{6}y(y + x(3x-5)(x-1))$ |
| 9 | $\frac{1}{6}y(y + x(3x+5)(x+1))$ |
| 10 | $\frac{1}{6}(x+1)^2(3xy+2y-1)$ |
| 11 | $\frac{1}{6}x(3(y-1)(x^2-y) + x(2y-1))$ |
| 12 | $\frac{1}{6}(y-1)(3x+1)(x^2-y) + \frac{1}{6}(y-1)(x+1)^2$ |
| 13 | $\frac{1}{6}y(-y + x(3x-5)(x-1))$ |
| 14 | $\frac{1}{6}x(3(y-1)(x^2-y) + x(-2y+1))$ |
| 15 | $\frac{1}{6}(x-1)^2(3xy-2y+1)$ |
| 16 | $\frac{1}{6}(y-1)(3x-1)(x^2-y) - \frac{1}{6}(y-1)(x-1)^2$ |

complicated than the one used in the case of the third order error functions: let us consider the function $c_{1111}(x, y)$ and suppose the function being approximated is of the form $\alpha x^4 + q(x, y)$, where $\alpha$ is constant and q is quadratic in x and y. Then (5.21) shows that the error function must be of the form $\beta x c_{111}(x, y) + \beta c_{1111}(x, y)$ for some constant $\beta$. Now the only fourth order term in the error in approximating a function of this form by a piecewise quadratic will clearly be the $x^4$ term; but we know that the only third order term in $c_{111}(x, y)$ is an $x^3$ term and therefore the only fourth order term in $c_{1111}(x, y)$ must be an $x^4$ term.

Figures 5.13, 5.14 and 5.15 give pictorial representations of the functions $c_{1111}$, $4c_{1112}$ and $6c_{1122}$ respectively. Figure 5.13 shows $c_{1111}(x, y)$ to be negative except at $x = \pm 1$, with a minimum value of $-^1/24$ at $x = 0$. $4c_{1112}(x, y)$ is zero round the boundary of the element and along the x and y axes, with zero gradient when $x = \pm 1$. The maxima and minima do not appear to be easily solvable analytically; numerical methods show that local maxima of approximately 0.0345887 occur at $\pm$ (0.4079010, 0.4472246), and the antisymmetry of the function means that minima of the same magnitude and opposite sign occur at $\pm$ (0.4079010, -0.4472246). Figure 5.15 indicates that the function $6c_{1122}(x, y)$ has a single minimum value of $-\frac{1}{4}$ at the origin; this function is zero with zero gradient along the boundary of the element and is negative everywhere inside the boundary.

In theory there is no reason why we cannot evaluate fifth, sixth and even higher order error functions. However there is little benefit to be gained from doing so: as the order increases the error functions become less important and more difficult to interpret; and of course they become extremely difficult to evaluate. We shall therefore conclude our error analysis having studied the third order error functions in some depth and having carried out a briefer examination of the fourth order corrections.

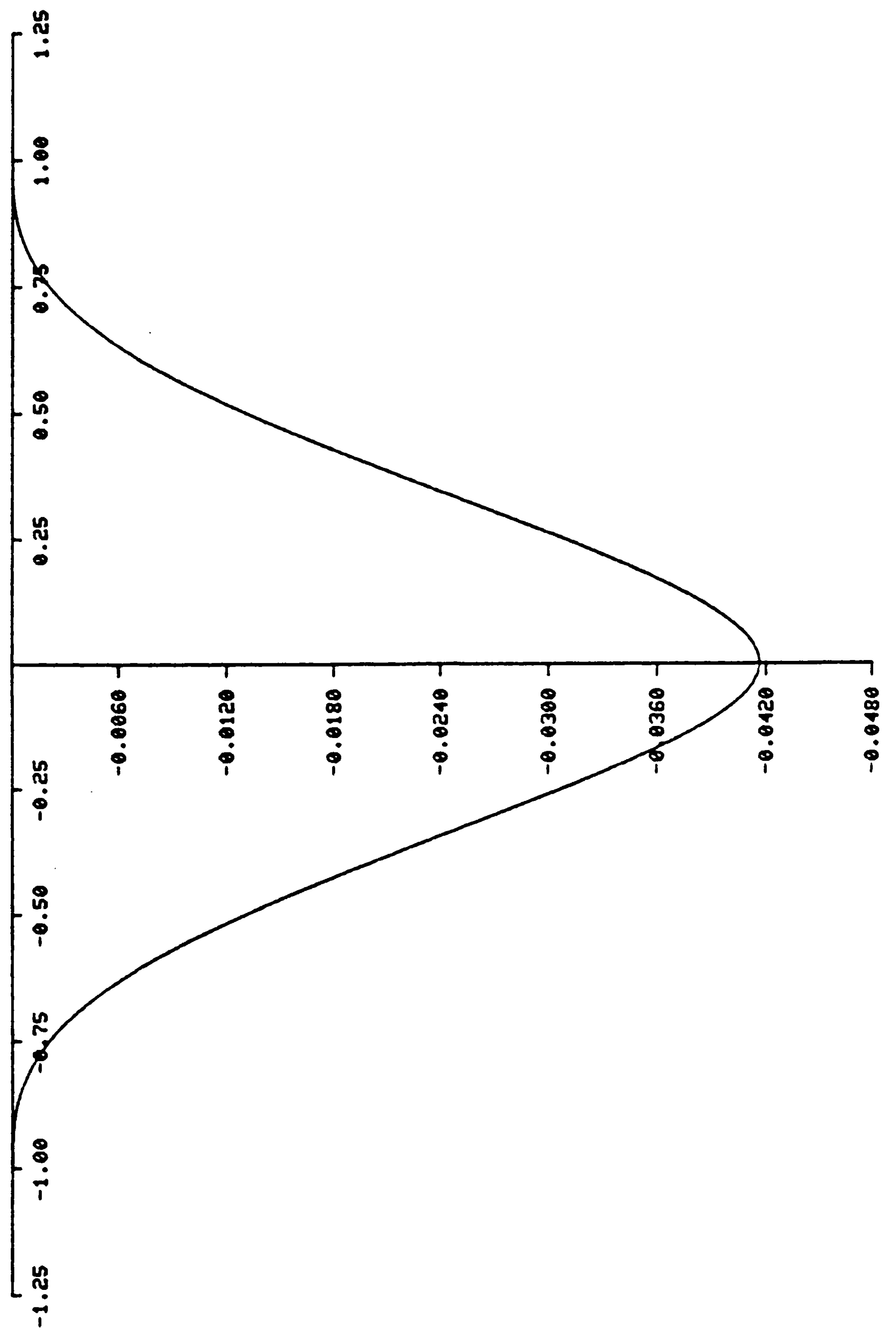


Figure 5.13 The fourth order error function $c_{1111}$ (x, y).

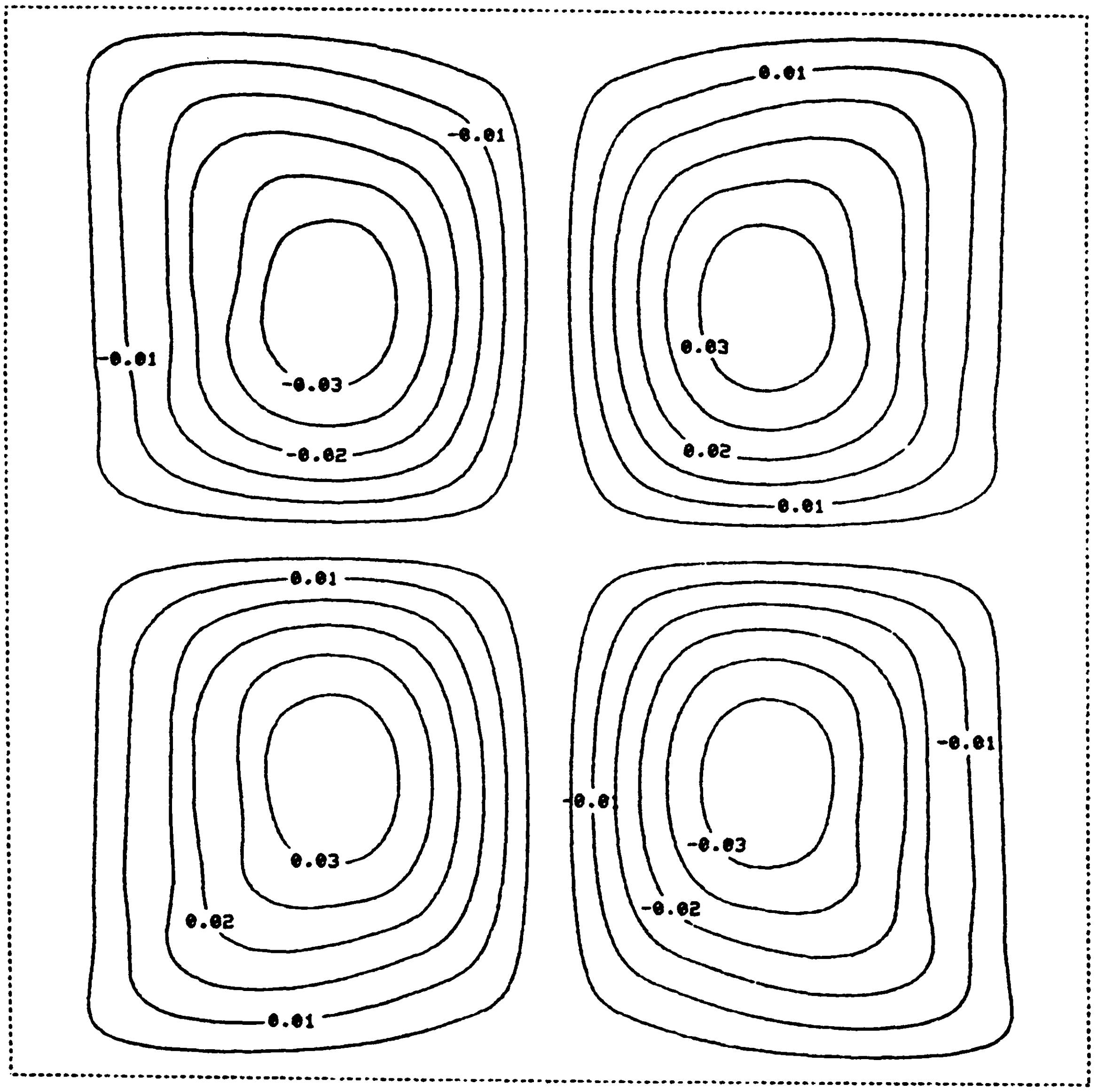


Figure 5.14 Contour plot of the fourth order error function $4\ c_{1112}\ (x, y)$.

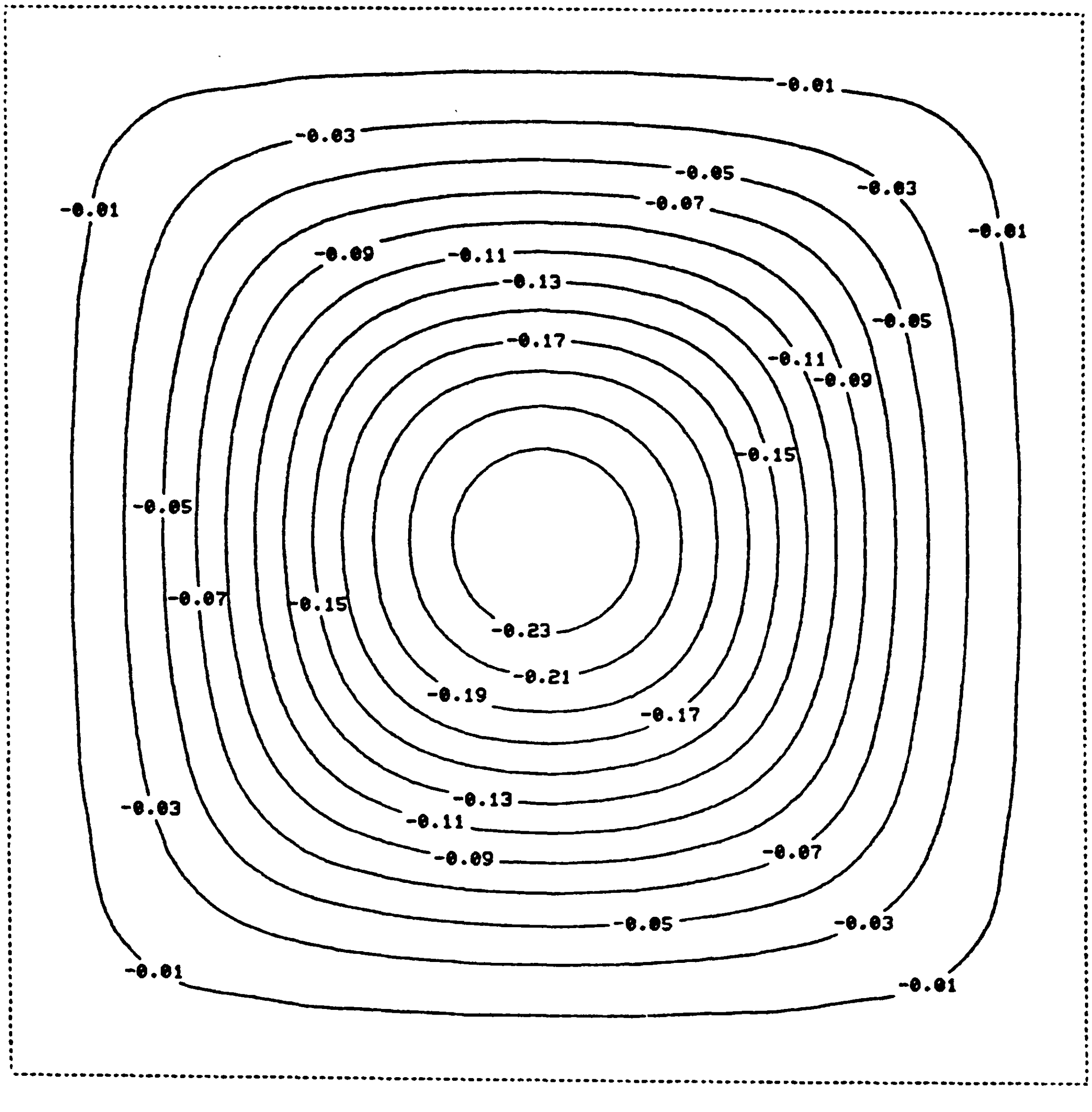


Figure 5.15 Contour plot of the fourth order error function $6\ c_{1122}\ (x,\ y)$.

# CHAPTER 6

# CONTOURING OVER LOCALLY ADAPTIVE GRIDS

## 6.1 Introduction

The idea of contouring across a grid with local variation in cell size in applications where value and gradient are calculable at any point was raised and discussed briefly in Chapter 5. The motivation for such a strategy stems from the observation that in most maps produced by CONICON, as the (uniform) grid size is decreased, the surface becomes well approximated in some areas of the plot long before it is adequately approximated in others; therefore, if a reliable indicator of goodness of approximation could be derived, we would prefer to carry out a local division of the grid cells confined to ill-approximated areas until the surface is well approximated everywhere. By doing this it ought to be possible to achieve considerable CPU savings ( by comparison with taking an everywhere fine grid) with no appreciable visible effect on the plot, and we should also make important savings in memory usage.

We therefore need to derive some suitable splitting rule to apply recursively to all cells within the grid. Many possible criteria for subdividing suggest themselves and some of these, based on maximum error etc, were considered in Chapter 5. In this chapter we shall test the performance of some of these ideas on known mathematical functions and we shall make further suggestions for splitting rules.

We begin though by considering the computational implementation of locally adaptive contouring, which is by no means a trivial problem to solve, and which of course is a prerequisite for further study of suggested splitting criteria.

## 6.2 Computational Implementation

### 6.2.1 Quad trees

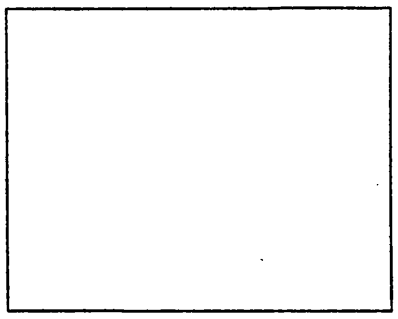
The particular sort of data structure which it is natural to associate with local splitting of grid squares is known as a quad tree; this is simply a tree whose nodes are either leaves or have four branches.

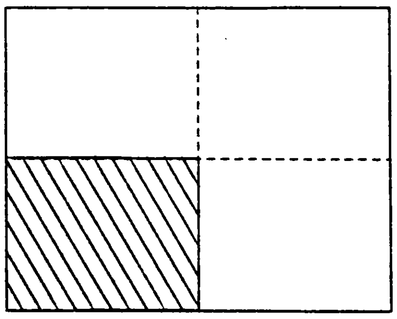
A review of the history of quad trees may be found in Klinger and Dyer (1976). In our case a leaf of the tree represents an element of the grid constructed in the manner explained above, and each node is associated with a quad, or square region of the plot which may or may not be subdivided into four smaller quads. Figure 6.1 shows an example of a typical grid structure and its associated quad tree (in which leaves are hatched).

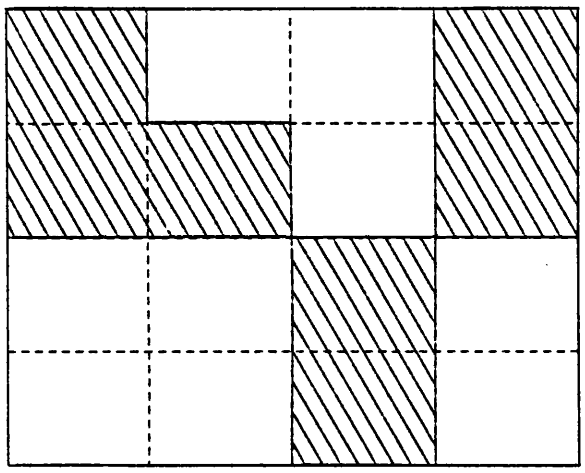
In fact if we make use only of data at the vertices of an element when considering whether or not it requires splitting (which will usually be the case if the function is fairly intractable analytically) we will probably construct a number of distinct quad trees, each emanating from a single cell of an initial coarse, regular grid. This will be done to lessen the likelihood (and the potential costs) of mistakenly failing to split an element because the data at the vertices happen erroneously to give the impression that the function is well approximated by the piecewise quadratic within the element.

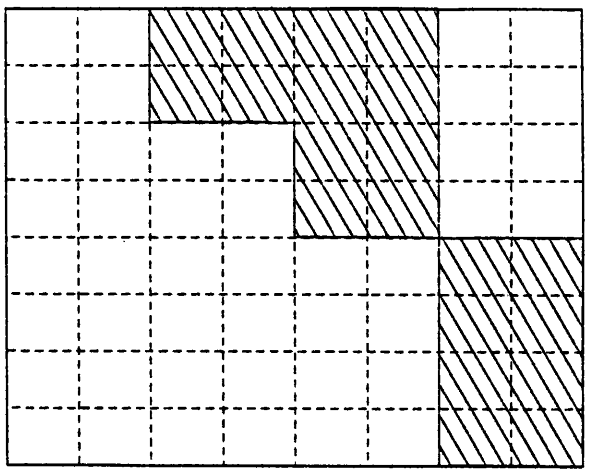
Thus we are likely to begin construction of the grid by selecting a regular grid of elements in such a way that we do not believe the behaviour of the function within any single element to be excessively complicated. It follows that each tree in our data structure will usually be a relatively small one, probably with at most three or four levels, and extremely unlikely to consist of more than five levels.

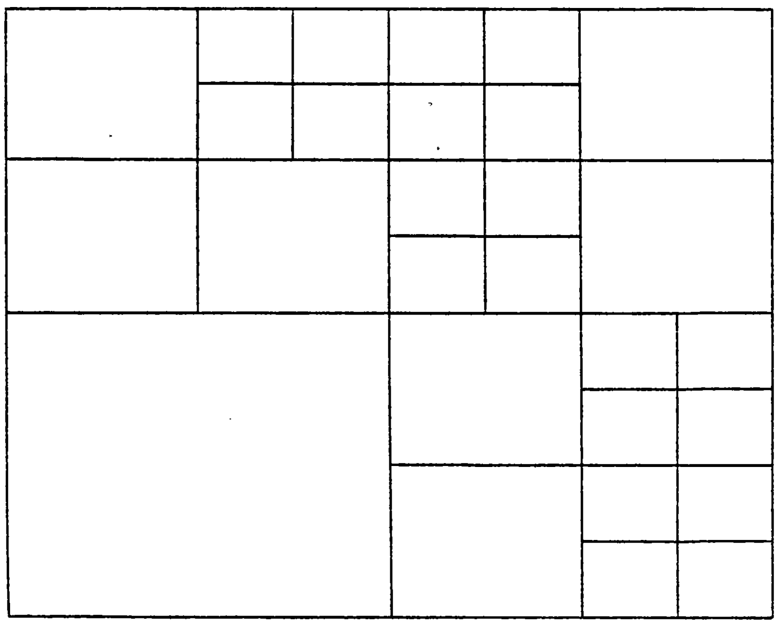
Figure 6.1   A typical grid structure and its associated quad tree.











Up to now the main use of quad trees has been as a memory-efficient method for picture storage on graphics devices (see for example Warnock (1969), Klinger and Dyer (1976), Woodwark (1982)). In such applications quads are subdivided until either the whole screen within an area can be represented by a single colour, or the resolution of the graphics device for which the picture is being prepared is reached. The greater the degree of coherence within the picture, the greater the saving in memory achieved using a quad tree. Quad trees have been developed extensively in such a way as to facilitate the various basic operations which are required in picture processing. In our case, however, few operations remain to be carried out once the tree has been set up, so we can dispense with most of the refinements which have been suggested to improve the execution of these operations.

There appears to exist two fundamentally different approaches to the problem of constructing a quad tree. The usual approach is to store a quad tree in the form of a linked tree structure, with links from a father to each of his sons, and sometimes additional links from a son to his father for back-tracking (see for example Klinger and Dyer (1976) or Hunter and Steiglitz (1979)).

The alternative, as proposed by Woodwark (1982) is to use an explicit or full quad tree, that is, a tree with storage locations assigned to every possible leaf which might occur in the tree, down to the smallest possible level where in the usual application individual leaves correspond to pixels. Woodwark argues that the absence of links in an explicit quad tree makes it more storage efficient than might be expected compared with a linked quad tree, particularly in cases where the number of bits required to store the data at each quad is small. However since, as we shall see below, a relatively large amount of information is stored at each quad in our case (and of course Fortran is too high-level a language to allow fine manipulation of the number of bits used at each quad) this argument is not very relevant to the contouring application. There is undoubtedly a significant saving in storage to be gained by using a linked tree structure.

Moreover, we can turn to our advantage a property that would normally be considered a disadvantage of using a linked tree structure: since the user has no precise knowledge of the final number of leaves in the tree he normally has to allocate significantly more storage than is necessary for the tree to ensure that the program does not fail. But in our case a program failure of this nature can indicate that the user has been too ambitious in the amount of accuracy demanded from his approximation, and it can save the user from carrying out a needlessly expensive contouring process and allow him to re-set his parameters.

Another reason advanced by Woodwark for the use of an explicit quad tree is the increase in speed of data input into and retrieval from the quad tree. In our case the former is not applicable, since no new data are input into the tree once its construction has been completed; and the effect of the latter is unlikely to be great, since as stated above, our quad trees are in general small in size and the time involved in processing links is therefore of no great order.

For the reasons outlined above it was felt that the case for using a linked tree structure was very strong in this application, and such a structure has therefore been adopted.

### 6.2.2 Coding policy for elements

Perhaps the most distinctive feature of the construction of the quad trees in our application is the need to determine the 'neighbours' of an element which we have decided to subdivide into four smaller elements. By 'neighbour', we mean an element which shares a common edge with the element being considered. The reason why we must locate an element's neighbours is explained below.

Suppose, for simplicity, that we are currently considering a neighbouring pair of elements A and B from the original regular grid of elements. After having considered whether each element of the grid needs to be subdivided we must evaluate a height and pair of gradients at every new node of the grid. Suppose a new node arises at the mid-point of the boundary line between A and B. This can occur for one of two reasons:-

(a) Both A and B need to be subdivided; or

(b) One of the elements (say A) requires subdivision and the other (B) does not.

These two cases require separate treatment: in the first case we simply evaluate the true height and gradients of the surface at the point of interest; however in the second case this is not possible, for the height and gradients at the midpoint of the boundary line are predetermined for us and are constrained to be the values taken by the piecewise quadratic on B at this point. To insert the true value and gradients of the surface at this point when contouring the two new elements bordering B on this side would result in a dis-continuity of the surface along the boundary line between A and B. Note though that no difficulties are caused by the fact that along each half of this boundary line we have a single quadratic on one side but a pair of quadratics on the other side: for the pair of quadratics is uniquely defined along the boundary line and so these two functions must be identical and the same as the quadratic on the other side of the line; also the condition of linearity of the normal component of the derivative ensures that this too is the same everywhere on both sides of the line.

Thus at this stage of constructing the tree we must first consider all elements of the grid in order to determine whether or not each one requires subdividing, and only then should we begin to determine the five new sets of data values required to complete the subdivision of those elements which are divided into four. One of these sets of values occurs in the centre of the parent element and can always be determined correctly. The other four lie on boundary lines with neighbouring elements from the original grid (unless they lie on the edge of the grid, in which case the true value and gradients of the surface may again be calculated); for each one of these we must locate the neighbour of the parent element in order to determine whether it too requires subdividing - if it does then we may determine the true value and gradients of the surface at this point (though we should be careful to avoid needless repetition of such evaluations by checking that the neighbouring element has not been dealt with already); otherwise we must evaluate the value and gradients of the neighbouring piecewise quadratic approximant at this point.

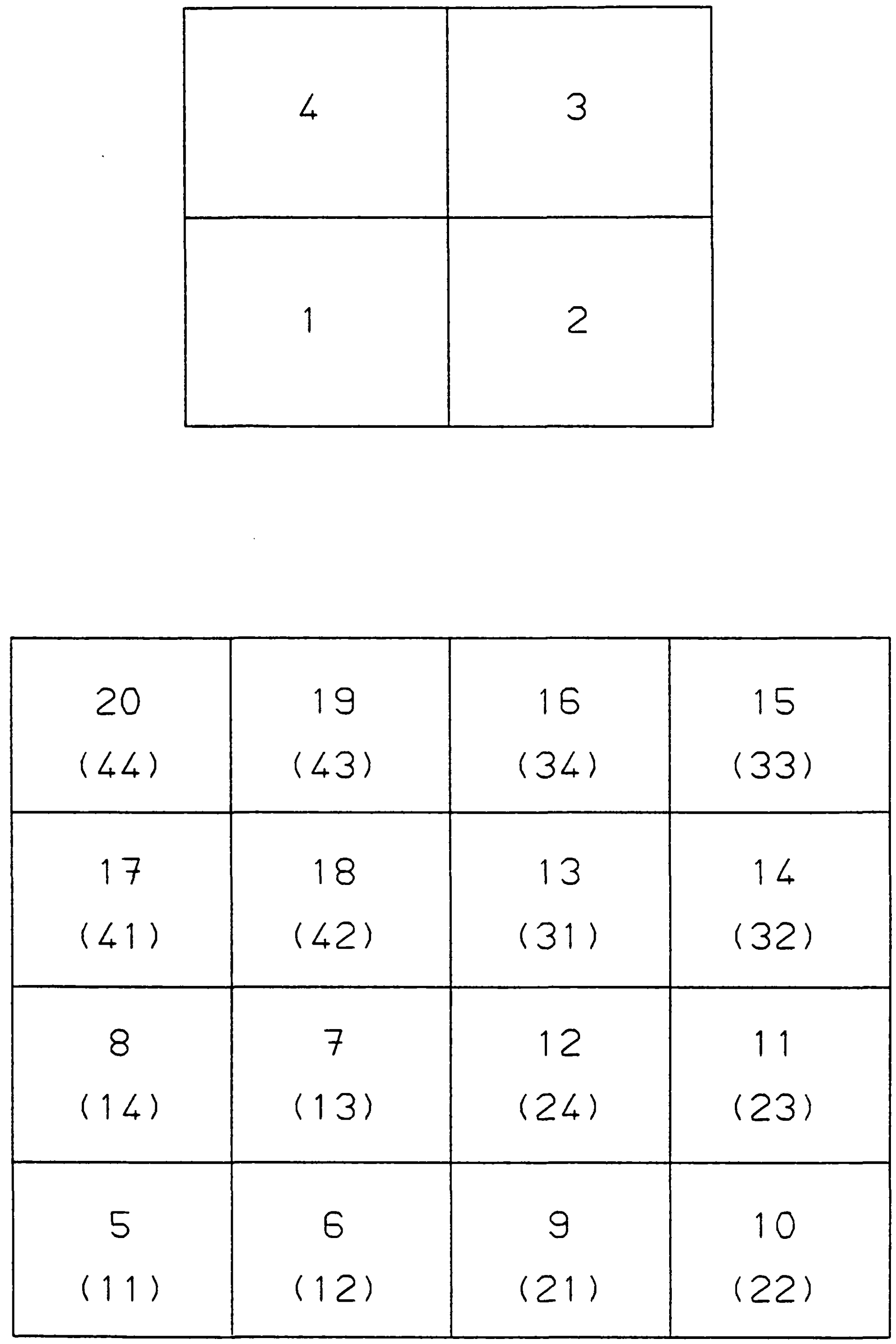
When we consider all quads at the next and subsequent levels of the tree, we proceed in a similar manner, but the process is slightly complicated by the fact that an element may not have a neighbour of the same size on one or two of its sides; this case however is resolved in exactly the same manner as it would have been if a neighbouring element of the same size did exist but did not require further subdivision.

It is therefore important to find a reasonably efficient method for locating the neighbouring element of the same size as the current element (if it exists) in any direction. This is a simple task if an explicit quad tree is used and quads are addressed according to their X and Y values, in the same way as a two-dimensional array might be addressed (such a policy is suggested - but not recommended - by Woodwark (1982)), but it would not be advantageous to implement a coding policy along such lines in the case of a linked tree because moving up or down the tree would become a major task, involving a substantial amount of computation.

The coding method employed in this application is therefore of a different nature (which does allow easy movement down the tree) and is illustrated by Figure 6.2. It has the advantage that once the code number of a neighbour has been established, information on the neighbour can be located reasonably quickly, because processing of links is a simple operation. Its disadvantage is that determining the code number of a neighbour is not as simple as the Cartesian coding system suggested above would allow. The figures in brackets in Figure 6.2 express the code number in base 4 (using digits 1-4 rather than 0-3 in order that each node of the tree at whatever level has a distinct code number). For any element in the grid (except those in the original regular grid) the code numbers of two of its neighbours (those two which share the same parent) may be determined by a simple alteration to the final digit. The code numbers of the other two neighbours are a little more difficult to determine.

Before describing the algorithm used to determine these code numbers, we shall introduce some notation and conventions.
Let NCODE denote the code number of the current element.
Let MCODE denote the code number of its neighbour.
NSIDE and MSIDE are both variables representing direction. The following conventions are employed:-

Figure 6.2 The second and third levels of a quad tree, showing the adopted coding policy.



|  |  |  |
|---|---|---|
| NSIDE = 1 | => | LHS or RHS |
| NSIDE = 2 | => | Above or Below |

|  |  |  |
|---|---|---|
| MSIDE = 1 | => | Below |
| MSIDE = 2 | => | RHS |
| MSIDE = 3 | => | Above |
| MSIDE = 4 | => | LHS |

To locate both neighbours of the element with code number NCODE which do not share the same parent (irrespective of whether they belong to the same element of the original regular grid), we repeat the following for NSIDE = 1 and NSIDE = 2.

(1) First decide which direction MSIDE we are looking in. Consider the final digit N of NCODE.
Then if NSIDE = 1,
N = 1 or 4 => MSIDE = 4; N = 2 or 3 => MSIDE = 2
or if NSIDE = 2,
N = 1 or 2 => MSIDE = 1; N = 3 or 4 => MSIDE = 2.

(2) Determine M, the final digit of MCODE, using the following:
If NSIDE = 1,
then M = MOD (6-N, 4) + 1
If NSIDE = 2,
then M = 5 - N.

(3) Find N, the next digit (working from back to front) of NCODE.
If N = 0, return. Otherwise use the formulae in (2) to determine M, the next digit in MCODE, and update MCODE.

(4) If MSIDE = N or MSIDE = MOD (N + 2, 4) + 1, go back to (3).

(5) The remainder of the digits in MCODE are the same as those in NCODE. Update MCODE to take account of this and return.

(6) End.

It was noted above that this algorithm returns the correct code value of the neighbour whether or not the neighbour lies in the same cell of the original grid. However, if the neighbour does not lie in the same cell we require some indication of this. Therefore, subroutine NENBR, which calculates the code number of a neighbouring element, returns the final value of N, which is the first digit of NCODE and the final value of M, which is MCODE's first digit. The neighbouring element belongs to a different cell of the original grid if any of the following occur:-

|  |  |  |
|---|---|---|
| MSIDE = 1 | and | N = 1, M = 4 |
|  | or | N = 2, M = 3 |
| MSIDE = 2 | and | N = 2, M = 1 |
|  | or | N = 3, M = 4 |
| MSIDE = 3 | and | N = 3, M = 2 |
|  | or | N = 4, M = 1 |
| MSIDE = 4 | and | N = 1, M = 2 |
|  | or | N = 4, M = 3 |

An efficient test is used to determine whether any of the above have occurred.

## 6.2.3 <u>Data structure used in adaptive contouring</u>

We now explain the data structure which is used in contouring over a locally adaptive grid.

The array VALS (3, IVTOP) replaces the arrays Z, ZX and ZY which are used when contouring across a regular grid. Each triple in VALS represents the height and partial derivatives of the surface (which may or may not be the true values) at a point. The data at any node are represented once only in this array.

The array IPTR (7, IQTOP) gives essential information on various quads of the tree, holding one set of seven values for every quad in the tree. If the quad is a leaf, then the first two values in IPTR supply the location of the element at the root of the current tree in the original regular grid, and the third value is the current element's code number within its particular quad tree. The final four values are pointers to the locations in VALS of the data at the SW, SE, NE and NW corners respectively of the current element. If the quad is not a leaf then values in IPTR will be exactly the same, except for the first value of the seven, which is flagged as negative to indicate this. In the present implementation no attempt is made to remove the data in IPTR which correspond to quads which are not leaves; some of this information is of use if we wish to plot the grid after the data structure has been set up, but values in the 4th to 7th dimensions are of no further use after the subdivision of an element is complete. Provision has however been made in the coding for these values to be overwritten in a future implementation as a means of saving memory: the amount of storage which might be said to be wasted through the retention of these negatively flagged values amounts to approximately one-seventh of the total storage used in IPTR.

Finally the array ITREE (ITOP) indicates the structure of the tree itself. The first (M-1) x (N-1) entries in ITREE each refer to a root of a quad tree. If the value corresponding to the Ith element is flagged as negative, then this element is not subdivided and data relating to it can be found in the -ITREE(I)th location in IPTR (though in the current implementation in such a case we will always have ITREE(I) = -I); otherwise if the value is positive the element is subdivided and the value points to the location in ITREE associated with the first son of the current element, information on the other three sons occurring in the three entries in ITREE immediately following this. After the entries in ITREE which correspond to the elements of the original grid, we have a similarly constructed series of entries relating to all quads at the next level down and then quads at lower levels, until we arrive at the final level where all values in ITREE are flagged with minus signs.

Note that no provision has been made for backtracking within the tree, as we have no need for it.

Subroutine ADGRID sets up the data structure described above. The user is required to supply a subroutine VALUES and to select a function DIVTST; the former should return the height and gradients of the true surface at a specified point, while the latter should determine, on the basis of the twelve data values at the vertices of an element along with its size and location, whether it requires further subdivision and return a value which is greater than unity if the element needs splitting and less than unity if no further subdivision is required. We describe below the major steps which subroutine ADGRID performs:-

1. Begin by calculating values and gradients on the original regular grid and adding these to the array VALS. At the same time initialise the relevant values in ITREE so that ITREE(I) = I and set appropriate values in IPTR for these elements.
2. Work through the list of elements of the current size in ITREE and for each one use DIVTST to determine whether it requires subdivision. If it does, flag the appropriate entry in the first dimension of IPTR as negative, reset the value in ITREE associated with this quad to point to the next free space in ITREE, and update the variable indicating the next free space in ITREE by making an addition of 4 to it. If the element does not require further subdivision, flag the appropriate value in ITREE as negative and calculate bounds for values taken by the surface within the element; store these in ZLIM (2, IQTOP).

3. The start of the main loop. Go back through the list of elements of the current size; if an element is a leaf of the tree then there is nothing further to do. Otherwise four extra sets of values need to be added to IPTR, corresponding to the four new elements, and up to five extra sets of values must be added to VALS. This process is carried out in Sections (4)-(7).
4. Calculate the code numbers of the new elements and place these, along with the 'coordinates' of the root of the tree, in the first three dimensions of IPTR.
5. Each of the four new elements has a vertex in common with its parent element, at which height and gradients have already been evaluated. Update IPTR to take account of this. Also each new element has a vertex at the centre of the parent element; call VALUES to evaluate the true height and gradients of the surface here, add these to VALS, and make appropriate additions to IPTR.
6. If the current element does not belong to the original regular grid, go forward to (7). Otherwise we know that all neighbours of the current element are of the same size, and information relating to them is easily located in IPTR. Consider each edge of the element in turn. For each edge, determine whether we are on the boundary of the plot: if so, evaluate the true height and gradients of the surface here and update VALS and IPTR accordingly. If the edge is not on the boundary of the grid, determine whether the neighbouring element is going to be subdivided or not. If not, calculate the height and gradients of the piecewise quadratic on the neighbouring element at the midpoint of the edge we are considering and add these to VALS before updating IPTR.

   If the neighbouring element does also require splitting then (a) if we are on the upper or right hand edge of the current element, evaluate the true surface height and gradients at the point of interest and update VALS and IPTR accordingly, not only updating the part of the latter which refers to two of the new elements within the current parent element, but also updating the part which is associated with the two new elements on the opposite side of the boundary line; or (b) if we are on the lower or left hand edge of the parent element this edge has already been dealt with, so go forward to the end of the loop.

When we have considered all elements within the original grid, go forward to (8).

7. When we arrive here the current element does not belong to the original grid, so we must carry out a more complicated process than the one described in section (6).

We follow through instructions (i)-(iv) first for NSIDE = 1 and then for NSIDE = 2 (i.e. first dealing with neighbours to the left and right, then with neighbours above and below).

(i) Call subroutine NENBR to determine the code number of the neighbouring element which does not share the same parent as the current one. Determine whether this lies within the same element of the original grid as the current element and, if not, decide whether we are on the boundary of the plot. If we are on the boundary, omit (ii).

(ii) Find whether a neighbour of the same size exists in this direction and, if so, whether it is also marked for subdivision.

(iii) Now carry out a process similar to that described in (6): if we are on the boundary of the grid, or the neighbouring element requires subdivision, evaluate the true data to place in VALS (as long as this has not been donealready) and update IPTR accordingly. Alternatively, if there is no neighbour of the same size in the current direction, or if such a neighbour exists but is a leaf of the tree, calculate the height and gradients at the appropriate point which will not disturb continuous differentiability of the approximant, add these to VALS and update IPTR.

(iv) Now deal with the opposite edge of the current element in a similar manner, the only difference being that the neighbour shares the same parent as the current element and therefore the location of its entries in IPTR can be determined simply without having to call NENBR or to process links of the tree.

8. If none of the elements of the current size have been subdivided, return. Otherwise update variables indicating the extent of information on elements of the current size in IPTR and ITREE, reduce the current grid size by a factor of two, and go back to (2).

### 6.2.4 Incorporation in the CONICON framework

The rather complicated process described above is quite straightforward to incorporate into the main framework for CONICON. To produce a contour plot using an adaptive grid, the user is required to call subroutine ADCON. This master routine is similar in many ways to subroutine ALLCON, which was described in Chapter 3: the major differences are that a call is made to ADGRID at an early stage in order to set up the grid and then instead of considering whether to call subroutine SQUARE once for each element of the regular grid, we consider each element which has a set of entries in IPTR. A further difference is that the bounds for values taken by the function over elements are not set up directly by subroutine ADCON, but during the execution of ADGRID.

From subroutine SQUARE downwards all routines used are exactly the same as those normally called in CONICON; it should be noted that in this case it is particularly important that subroutine JOIN does not attempt to match segment ends exactly, because at the boundary of a pair of elements of different sizes it is particularly difficult to calculate contour intersections on both sides in exactly the same manner.

Crosshatching over adaptive grids has not been implemented by the author; nor has contouring within a non-rectangular window or gradient plotting, though these two problems can be treated in exactly the same way as previously, and something very similar to 'algorithm A' could be used successfully for crosshatching. The only major alteration here would appear to be in determining the height of the surface on the boundary of the grid and this does not appear to be particularly difficult to implement.

It is clearly of the utmost importance to have some means of plotting the grid over which we are contouring and, as mentioned above, a subroutine, called PLTGRD, has been written for this purpose. This subroutine first plots the regular grid on which the construction is based, and then looks through the array IPTR for nodes of the tree which have been flagged as negative i.e. elements which have been sub-divided to form four smaller elements. Whenever the routine comes across such an element it plots the division which has occurred within it. After all data in IPTR have been dealt with in this way the plot is complete.

The author is aware that there is some scope for improvement in the method described above for setting up the data structure for adaptive contouring, both in terms of running time and memory used. However, experimental results have shown that the time taken to set up the grid in most examples is fairly insignificant in comparison with the time taken to contour over the grid, and it is therefore felt that efforts to improve the process of setting the grid up are only marginally justifiable, as they could not possibly result in any significant improvement in the total job execution time.

In the following sections in this Chapter we see several examples of the use of a locally adaptive grid for contouring which use the implementation described in this section.

## 6.3 Splitting criteria based on maximum vertical error

### 6.3.1 Introduction

In Chapter 5 we derived a bound for the maximum (absolute) error involved in using the seamed quadratic element to approximate a function with bounded third derivates, the upper limit being a simple function of the maximum of the four third order partial derivates of the true surface over the area of the plot. We suggested that this bound might be used as the basis of a criterion for local splitting of grid elements, in such a way that subdivision of an element should occur if and only if the bound for error within that element exceeded a specified value. So long as this prescribed value was positive, reasonably fast termination of the process would be guaranteed for all but the most ill-behaved functions, since the bound would decrease by a factor of approximately 8 with each splitting process carried out.

In this section we test the performance of such a criterion on a pair of mathematical functions. The first is the mixture of two bivariate Normal distributions which was introduced in Chapter 1 and has been used as a test example on a number of occasions in previous chapters. The function is

$$f_1(x, y) = \exp(-\tfrac{1}{2}(4(x-1)^2 + 6(y-1)^2))$$

$$+ \exp(-\tfrac{1}{2}(10(x-2)^2 + 6(y-1.3)^2+14(x-2)(y-1.3))) \qquad (6.1)$$

The second test function is

$$f_2(x, y) = \exp(-br)\cos(cr) \qquad (6.2)$$

where $r = \sqrt{(x^2+y^2)}$; $b = 0.06$; $c = 2\Pi/3$

The latter function is particularly difficult to contour; it may be regarded as a series of equidistant ripples of exponentially decreasing magnitude emanating from the origin. The function is differentiable everywhere except at the origin and its contours are circles centred on the origin. Figure 6.3 shows a contour plot of this function using a regular grid of 25 x 25 points. The contour heights are at regular intervals and the surface has been contoured over the interval [0.25, 12.25] x [0.25, 12.25] in order to avoid discontinuity problems. The contours in Figure 6.3 appear quite satisfactory, but when contour levels approach any of the areas where the contribution of the cosine function is at a maximum or minimum and the contours coincide, behaviour is bound to be less satisfactory. Two such levels occur when $r = 12$ and $r = 7.5$. These correspond to function values of approximately 0.48675 and -0.63763 respectively, and Figure 6.4 shows the behaviour of a pair of contours of the piecewise quadratic approximant close to these levels. Such behaviour can never of course be completely eliminated using our contouring method, but we might hope to obtain some degree of local splitting of the elements in the appropriate areas to improve matters.

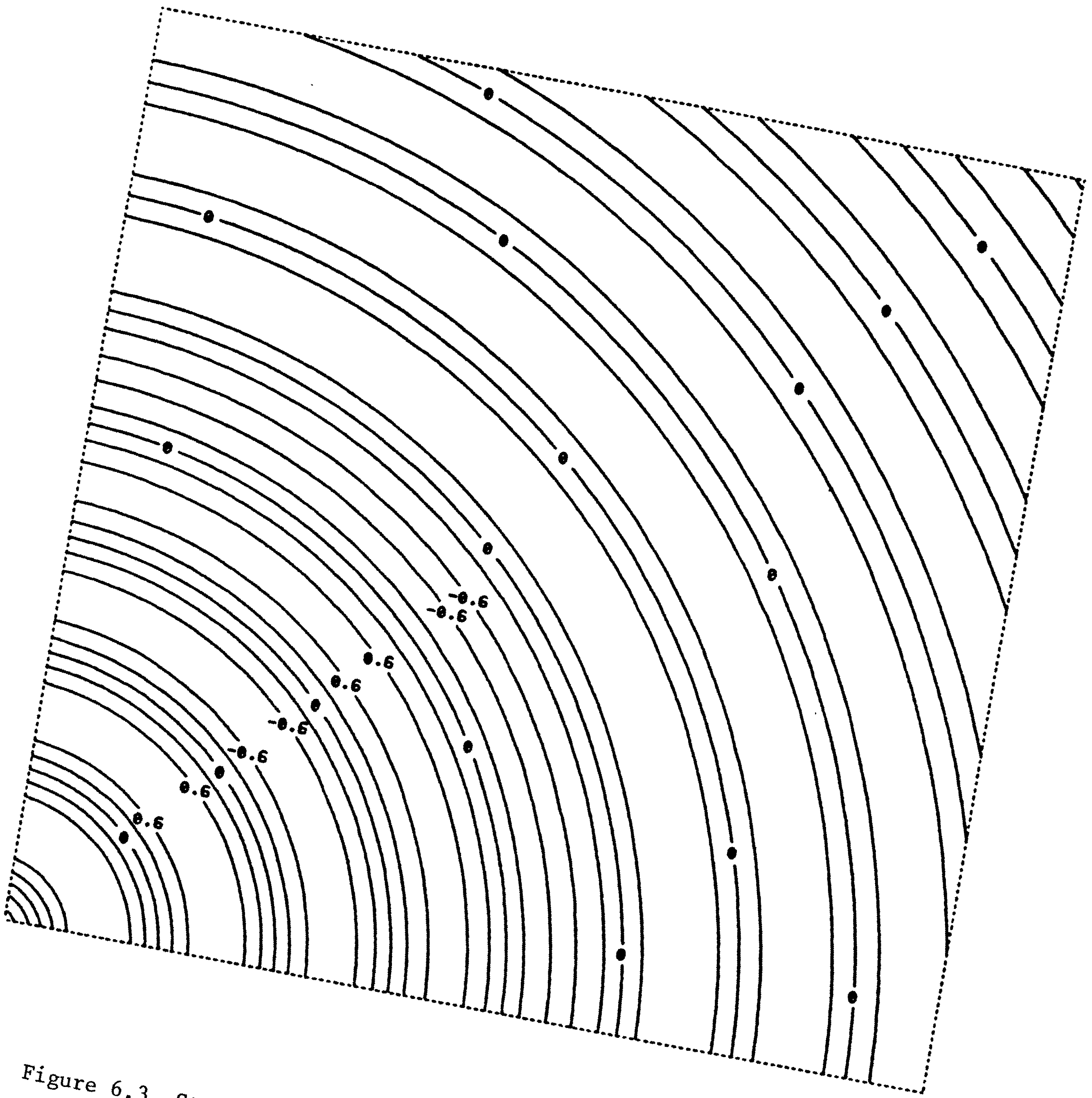


Figure 6.3 Standard contour plot of function $f_2$ (x, y).

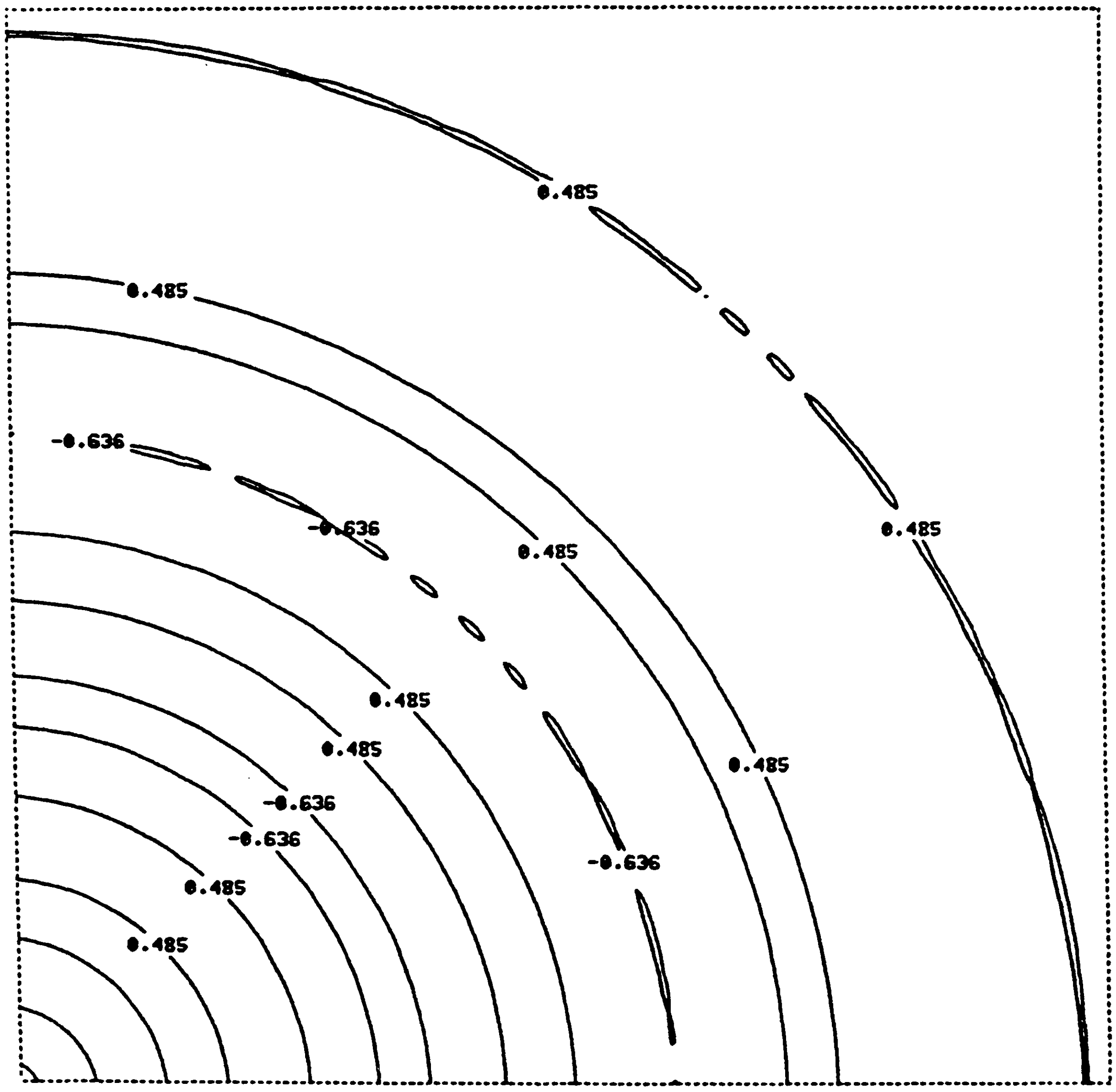


Figure 6.4 Standard contour plot of function of $f_2$ (x, y), showing behaviour close to a local minimum and and a local maximum of the cosine function.

## 6.3.2 Use of true third order partial derivatives

The suggested splitting criterion requires us to evaluate the maximum magnitude of any of the function's third order partial derivatives within each element of the grid. This is not really a practicable exercise with either of our test functions so instead, as suggested in 5.4, we opt for the alternative of choosing the largest third order partial derivative at any of the element's vertices. Note that this is sometimes liable to result in underestimating the third derivative, but if the function is well-behaved this should not happen very often and the effects will not be very serious when it does happen.

For function 1, if we evaluate all third order partial derivates on the 21 x 31 regular grid of points used in the production of Figure 2.5, we find that the maximum value occurs at (1.9, 1.1), close to the peak of the distribution in which the x- and y- variables are correlated, and is approximately 43.3327. Using the bound for error given by Theorem 5.1, this tells us that the maximum error involved in using the piecewise quadratic approximant to approximate function 1 over this regular grid is almost certainly bounded above by 1.07e-03 (since $h = 0.05$). However if we use the criterion suggested and choose the grid size adaptively, we can use a smaller number of elements and still ensure (almost) that the error in approximation is always less than this value, or alternatively we can use a similar number of elements to achieve a reduction in the bound for maximum error.

In Figure 6.5 we have chosen the latter option; in fact we have 612 elements compared with 600 elements in Figure 2.5. The error in approximation in this example has been reduced to 1.0e-03, perhaps not as large a reduction as we might have hoped for, but this is probably a result of the smooth, well-behaved nature of the function. The

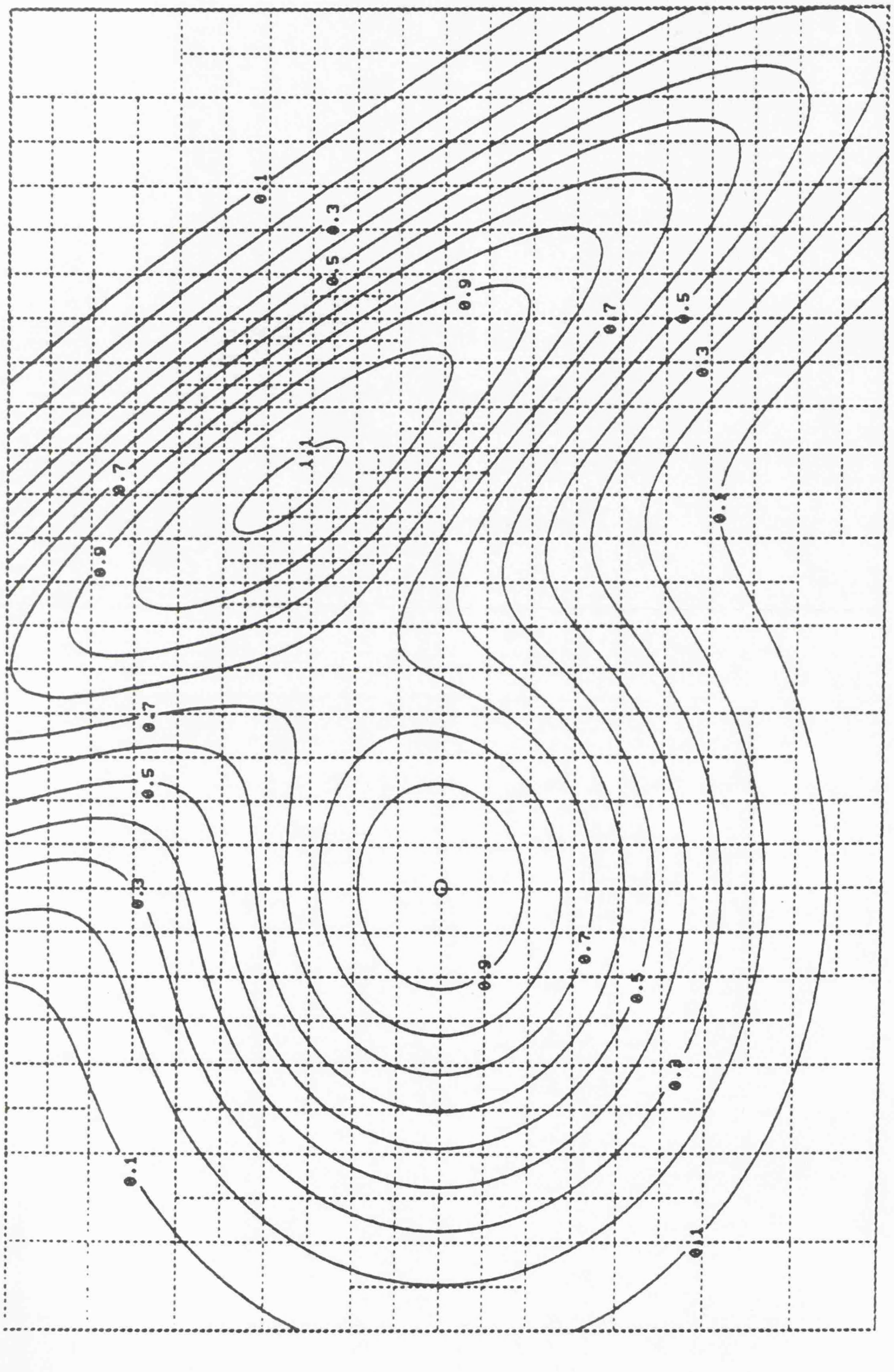


Figure 6.5 Function $f_1$ (x, y). Splitting based on largest third order partial derivative at any vertex.

pattern of elements in Figure 6.5 is encouraging nevertheless: over most of the plot elements are of a uniform size, but near the edge of the plot where there are few contours and the surface is relatively flat they are larger, while around the right hand peak (which Figure 2.6 shows to be a particularly difficult area to contour) we have a large concentration of small elements.

We now move on to consider function 2. In this case the maximum value of all third order partial derivatives evaluated at the nodes of the 25 x 25 grid of points used to produce Figures 6.3 and 6.4 is approximately 8.4985, signifying a maximum error of less than 2.623e-02. If we use our splitting criterion (beginning with a 6 x 6 grid of elements) in an attempt to produce a locally adaptive grid with a similar maximum error we find that there is very little difference in the grid pattern, with just a few elements of size h = 0.5 remaining undivided near the top right hand corner of the plot. In order to study the behaviour of this particular splitting criterion on this function we have therefore selected a maximum error of about half the size, namely 1.3e-02, which resulted in the pattern of elements indicated by Figure 6.6.

Probably the most noteworthy feature of this illustration is the tendency of the smaller elements to aggregate close to the x- and y-axes, indicating an obvious failing of the method, for we would like the method to be reasonably invariant under rotation of the axes, given an examples of this nature. We cannot expect total invariance under rotation of the axes, because both the partial derivatives and the grid itself are not invariant under rotation, and for the latter reason in particular we are bound to obtain significant departures from optimum results with a function of this complexity.

However what is perhaps surprising is the scale of the departure from invariance under rotation, and this gives cause for concern. An

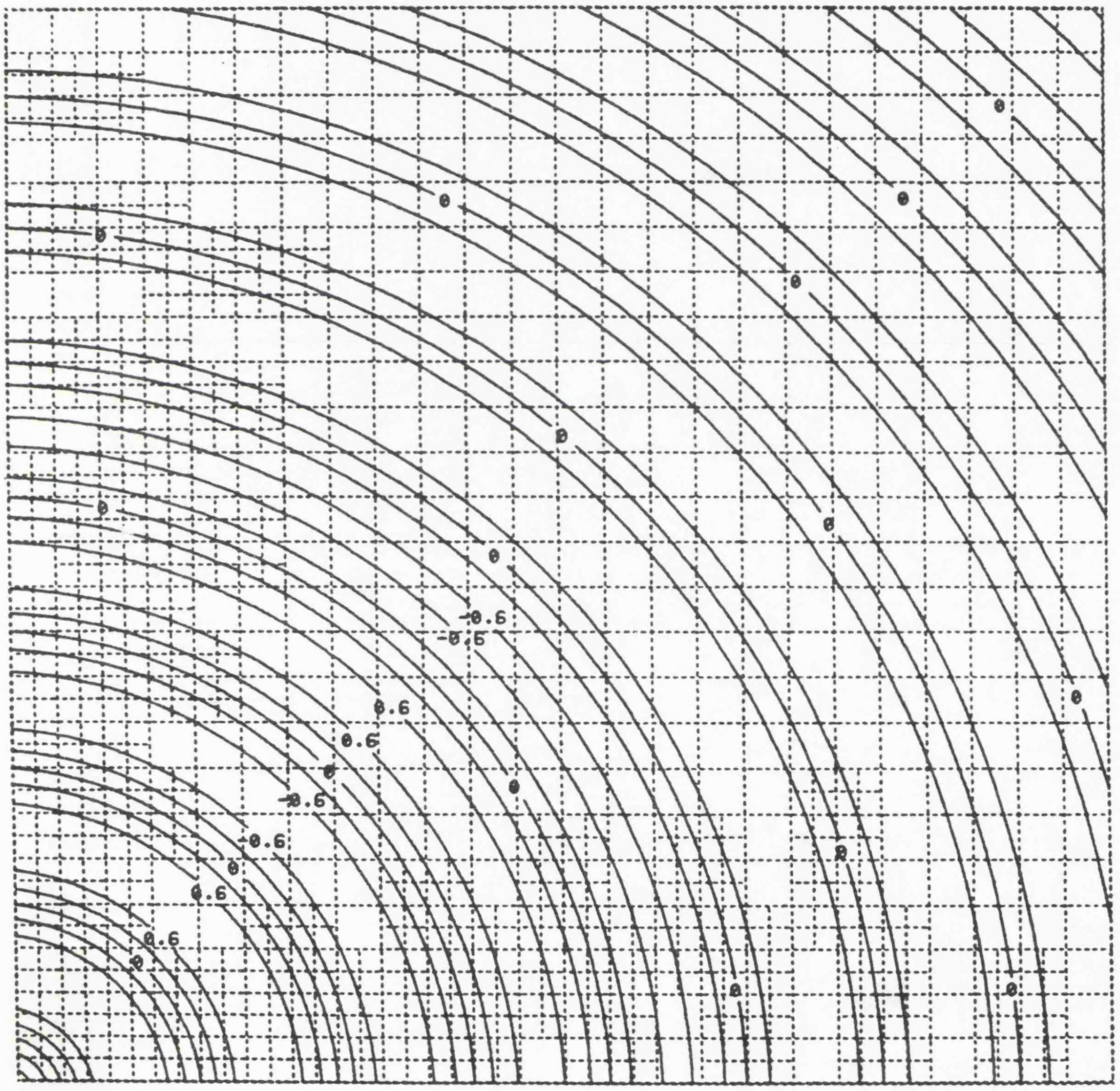


Figure 6.6 Function $f_2$ (x, y). Splitting based on largest third order partial derivative at any vertex.

improvement would undoubtedly arise if it were practicable to calculate the true maximum of the third order partial derivatives within each element; note that another consequence of only considering derivates at the vertices of the element is that any element which has been subdivided is necessarily part of a block of four such elements forming a large square.

Finally it should be noted that the grid pattern indicated by Figure 6.6, although possessing a considerably larger number of elements than the regular grid used to produce Figure 6.4, would have little effect on improving the worst contours in Figure 6.4, which tend to occur in areas some distance away from the axes.

## 6.3.3 Estimating third order partial derivaties

In a number of cases it is possible to evaluate the third order partial derivatives of our function at the vertices of an element; however this may in some examples not be achieved without considerable labour (as well as expense for the computer) and the prospect of having to carry out such calculations might well deter many potential users from attempting to use the adaptive method for contouring. Indeed, computation of the third derivatives of our two test functions is by no means a trivial task. Therefore, as the emphasis of this work is on developing methods which are widely applicable and can be handled by scientists from diverse disciplines, we would prefer to be able to offer a method which does not require the user to evaluate third order derivatives of the functions but instead automatically estimates these quantities in some (we hope) reliable manner. It would be particularly useful if a reasonable estimate of the maximum value of the third order partial derivatives could be derived solely from the twelve data values at the vertices of an element; such an estimate is developed below.

We begin by considering an element on the 'unit square', as defined in Chapter 5, with vertices at $(\pm 1, \pm 1)$. It is stressed though that what follows is completely independent of the nature of the element and is simply a consequence of the locations of its vertices. If value and gradients are evaluated at each vertex of the square then we have twelve pieces of information regarding the function we are contouring. It therefore does not seem unreasonable to hope that from these data values we could determine all ten parameters of a cubic polymomial if our surface was such a function, and indeed this is possible. In fact we can do still better than this: if we introduce a pair of quartic terms into our polynomial, namely an $x^3y$ term and an $xy^3$ term, the coefficients of these terms are also identifiable. Suppose however that the function which we are contouring is not a twelve parameter function, but a general quartic with fifteen parameters; it will no longer be possible to identify all the parameters of the polynomial or indeed all the twelve parameters which we have just considered, since some of these will be confounded with the parameters corresponding to the coefficients of the three additional quartic terms. However, by good fortune, we find that all the cubic coefficients as well as the coefficients of the $x^3y$ and $xy^3$ terms are still identifiable, so we can still obtain a considerable amount of information regarding the third derivative of our function. The following shows why this is true. Consider the functions,

$$\phi_1(x, y) = (x+1)^2(x-1)^2 = x^4 - 2x^2 + 1$$

$$\phi_2(x, y) = (y+1)^2(y-1)^2 = y^4 - 2y^2 + 1$$

$$\phi_3(x, y) = (x+1)(x-1)(y+1)(y-1) = x^2y^2-x^2-y^2+1 \qquad (6.3)$$

or alternatively replace $\phi_3(x, y)$ by

$$\phi_3'(x, y) = 2x^2y^2-x^4-y^4$$

The three functions given by (6.3) are linearly independent and have the following property in common; if the function $\phi_i(x, y)$ and its first order partial derivatives are evaluated at each of the four vertices of our square, all twelve values and gradients are zero. Thus any multiple of one or more of these functions could be added to the function being contoured with no effect whatsoever on the values and gradients at the vertices of the unit square and consequently with no effect on the contours plotted. Therefore the effect of adding the other three quartic terms to our twelve-parameter polynomial is to render the coefficients of the six terms $x^4$, $y^4$, $x^2y^2$, $x^2$, $y^2$ and 1 unidentifiable. The remaining nine parameters (which include all cubic and two quartic terms) can still be identified and indeed they are quite simple to calculate.

The expressions derived for the six particularly useful coefficients are of some interest, because they are closely connected with the idea of tangent intersections which we discussed in Chapter 2. Recall that the tangent intersection property of a one-dimensional quadratic function states that tangents constructed at any two points $x_1$ and $x_2$ on the line have a common value at the point $\frac{1}{2}(x_1+x_2)$. Thus we might conjecture that if we have a function which is not quadratic with tangents constructed at points $x_1$ and $x_2$, the difference between the values of these tangents at $\frac{1}{2}(x_1+x_2)$ could perhaps be considered to be a suitable measure of the departure from quadraticity of the function. Indeed we find that the six identifiable parameters of our quartic which contribute to the third derivative are all functions of such quantities, measured along the edges and the diagonals of our element.

Suppose $a_{ij}$ is the coefficient of the $x^iy^j$ term in our polynomial. Suppose also that we use the following notation to describe the difference between tangent values at the midpoints of edges and diagonals of our element (now of dimension 2h x 2h):

$$T_S = (z_{SW} + hs_{SW}) - (z_{SE} - hs_{SE})$$

$$T_N = (z_{NW} + hs_{NW}) - (z_{NE} - hs_{NE})$$

$$T_W = (z_{SW} + ht_{SW}) - (z_{NW} - ht_{NW})$$

$$T_E = (z_{SE} + ht_{SE}) - (z_{NE} - ht_{NE})$$

$$T_{SW,NE} = [z_{SW} + h(s_{SW} + t_{SW})] - [z_{NE} + h(-s_{NE} - t_{NE})]$$

$$T_{SE,NW} = [z_{SE} + h(-s_{SE} + t_{SE})] - [z_{NW} + h(s_{NW} - t_{NW})] \qquad (6.4)$$

Then the identifiable cubic and quartic coefficients are as follows:

$$a_{30} = (T_S + T_N) / 8h^3$$

$$a_{21} = (T_{SW,NE} + T_{SE,NW} - T_W - T_E) / 8h^3$$

$$a_{12} = (T_{SW,NE} - T_{SE,NW} - T_S - T_N) / 8h^3$$

$$a_{03} = (T_W + T_E) / 8h^3$$

$$a_{31} = (T_N - T_S) / 8h^3$$

$$a_{13} = (T_E - T_W) / 8h^3 \qquad (6.5)$$

The other identifiable parameters are:

$$a_{11} = (z_{SW} - z_{SE} + z_{NE} - z_{NW}) / 2h^2$$

$$+ (s_{SW} + s_{SE} - s_{NE} - S_{NW} + t_{SW} - t_{SE} - t_{NE} + t_{NW}) / 8h$$

$$a_{10} = 3(-z_{SW} + z_{SE} + z_{NE} - z_{NW}) / 8h$$

$$+ (-s_{SW} - s_{SE} - s_{NE} - s_{NW} - t_{SW} + t_{SE} - t_{NE} + t_{NW}) / 8$$

$$a_{01} = 3(-z_{SW} - z_{SE} + z_{NE} + z_{NW}) / 8h \quad +$$

$$(- s_{SW} + s_{SE} - s_{NE} + s_{NW} - t_{SW} - t_{SE} - t_{NE} - t_{NW}) / 8 \qquad (6.6)$$

and if the coefficients of $x^4$, $x^2 y^2$ and $y^4$ are zero, then

$$a_{20} = (- s_{SW} + s_{SE} + s_{NE} - s_{NW}) / 8h$$

$$a_{02} = (- t_{SW} - t_{SE} + t_{NE} + t_{NW}) / 8h$$

$$a_{00} = (Z_{SW} + Z_{SE} + Z_{NE} + Z_{NW}) / 4$$

$$+ h(s_{SW} - s_{SE} - s_{NE} + s_{NW} + t_{SW} + t_{SE} - t_{NE} - t_{NW}) / 8 \qquad (6.7)$$

(This final expression has appeared previously as (2.16)).

We may use the six identifiable coefficients of third and fourth order terms to estimate the maximum third order partial derivative over an element: if we approximate the true function being contoured by a twelve parameter polynomial of the type described above we can evaluate the maximum third order partial derivative of this function over the element and use this as our estimate of the maximum third order partial derivative of the true function. As each of the four third order partial derivatives of our polynomial is linear in x or y only, the maximum value will occur along one of the edges of the element and is easily calculated.

We now return to our two test functions and employ this third derivative estimate as the basis of our splitting criterion:- in the case of function 1, if we first evaluate the estimate on each cell of a 20 x 30 regular grid of elements we find that a maximum of approximately 42.0547 occurs within the element covering the region [1.9, 2.0] x [1.0, 1.1], a result which is encouragingly close to the maximum of approx. 43.3327 found at (1.9, 1.1) when we measured the true derivatives at vertices only. We have found that in general this new estimate of third order partial derivative tends to understate the value calculated from third derivative measurements at the vertices, which in turn of course is sometimes less than (and never greater than) the true maximum of the third derivative of the surface within an element. Thus Figure 6.7, which employs our estimate in combination with the bound given by Theorem 5.1 and was the result of requesting a maximum error of 8.0e-04 (c.f. 1.0e-03), uses only a slightly larger number of elements (651 compared with 612) than Figure 6.5.

However the pattern of splitting of elements is broadly similar to the one achieved in the previous subsection, suggesting that the estimate derived above might well provide a suitable criterion for splitting of elements in most examples, though it seems unlikely that it could be used to obtain anything more than a very rough impression of the accuracy of approximation.

Next we consider the performance of our new splitting criterion on the second test function. On this occasion the estimated maximum third order partial derivative using a regular 24 x 24 grid of elements is 6.99385, significantly less than the value of around 8.5 which we arrived at using the method discussed earlier. The evidence of plots of function 2 produced using this criterion suggests that the high order of the difference is not just confined to the area near the origin where the maximum third derivative occurs, but throughout the area of the plot. Figure 6.8 shows a grid pattern produced using the new criterion which is comparable with Figure 6.6, having only 60 more elements. This was produced by setting the requested maximum error to 1.0e-02 (c.f. 1.3e-02 for Figure 6.6). It can be seen that the two patterns of elements are again broadly similar, but if there is anything to choose between the two then the grid shown by Figure 6.8 appears more successful than that illustrated by Figure 6.6: the smaller scale elements are not quite so closely aggregated near the axes and they tend to

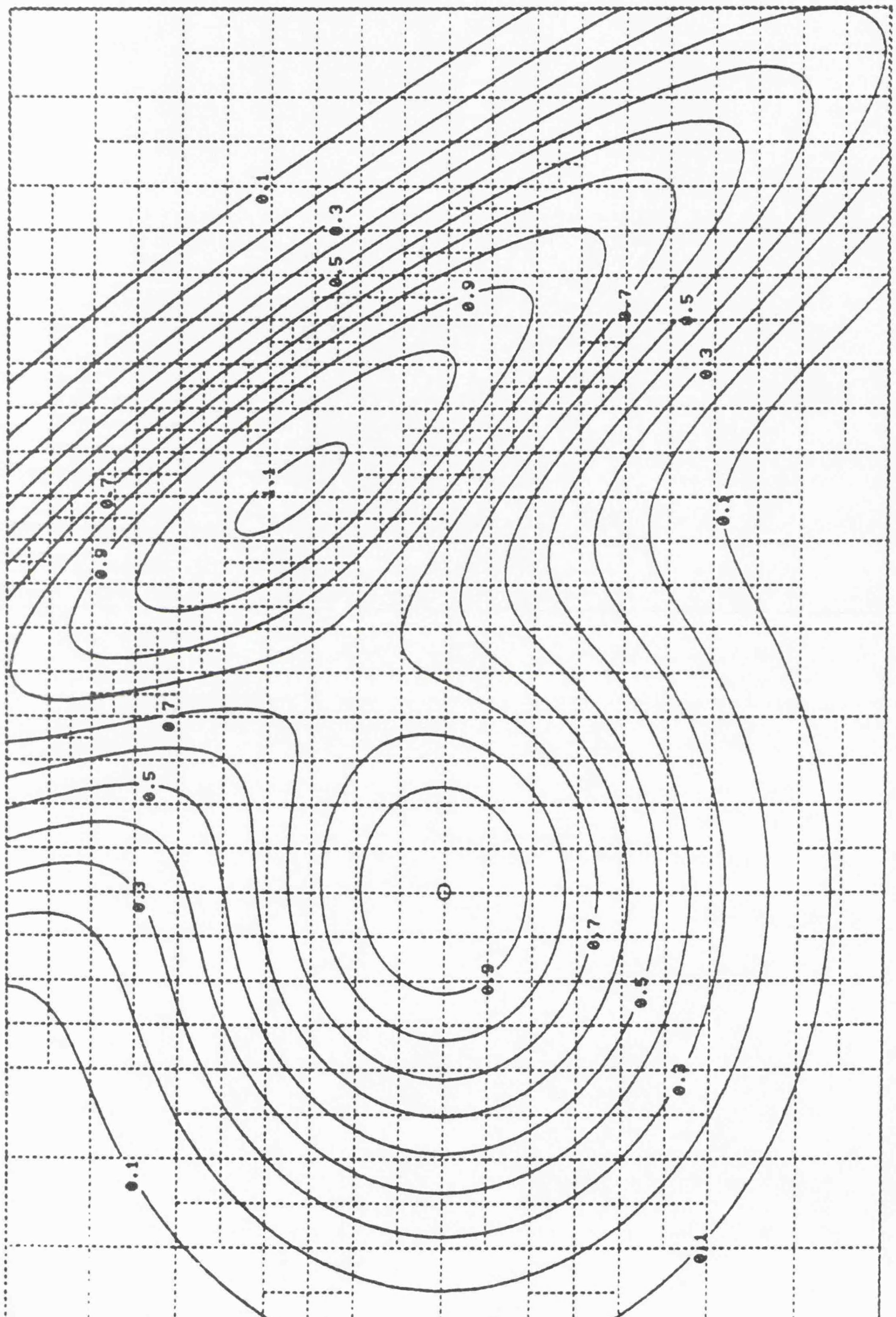


Figure 6.7 Function $f_1$ (x, y). Splitting according to estimated maximum third order partial derivative within an element.

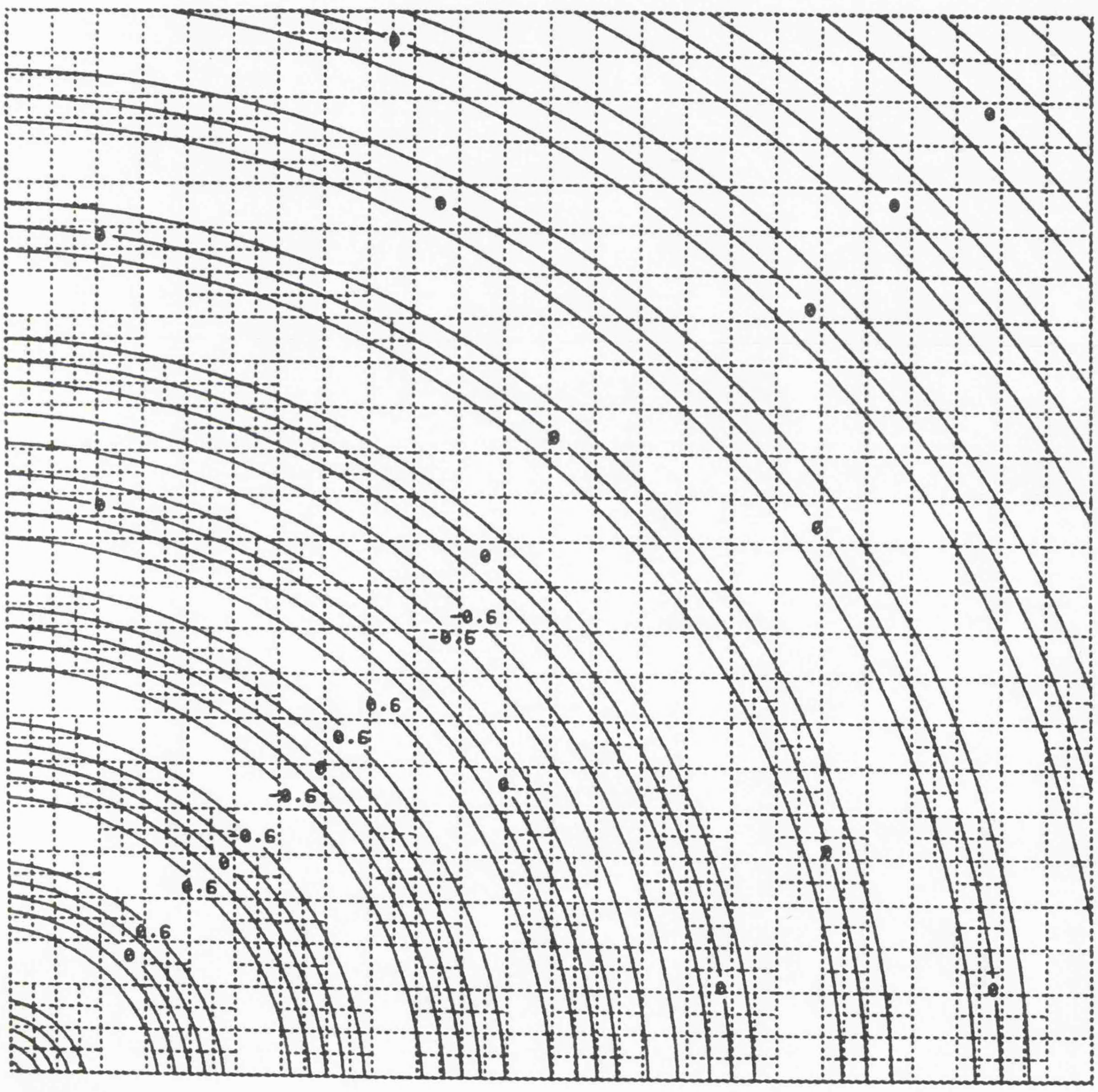


Figure 6.8 Function $f_2$ (x, y). Splitting according to estimated maximum third order partial derivative within an element.

occur in areas where the density of contours is high; in other words in areas where the gradient of the function is large and we would expect a relatively large error in approximation. Of course it is not in such areas that the worst contours of Figure 6.4 occur, but as we are considering vertical error only in this section this is to be expected. In the following section we consider horizontal rather than vertical error in an attempt to improve contour behaviour in these areas.

From the results obtained in this section we may conclude that when maximum vertical error is used as the basis of a criterion for local splitting of elements, the estimates of third derivative derived above from the twelve data values at the vertices of an element lead to a promising looking criterion for local division of grid squares which is also simple to operate.

## 6.4 Splitting criteria based on horizontal error

### 6.4.1 Introduction

In this section we consider the performance of a splitting criterion based on horizontal rather than vertical error on the pair of test functions introduced in the previous section. The concept of horizontal error and arguments for and against its usefulness were discussed at length in Section 5.4 and we shall not dwell further on these arguments any more than is necessary in this chapter. In 5.4 we also introduced three potential measures of flatness of the surface within an element and suggested a way of combining any one of these measures with some estimate of maximum vertical error in order to derive a suitable indicator of the degree of horizontal error of the contours within an element. Although the use of one of these three measures was tentatively recommended in preference to the other two, no tests were carried out to examine how well each behaved. We therefore begin with a brief comparison of these measures.

In order to compare the measures of flatness we consider once again the piecewise cubic polynomial which was plotted as Figure 5.6. With the exception of the diagonals of the plot, where it is only once differentiable, this function has a constant maximum (absolute) third order partial derivative of 1.0 throughout its domain. Recall that we defined our suggested measure of horizontal error to be

$$\frac{\text{maximum vertical error}}{\text{measure of flatness} + \kappa} \quad (\kappa > 0)$$

It follows that if we were to construct a locally adaptive grid for plotting this particular function using the above measure as splitting criterion, subdivision of elements would depend only upon the measurements of flatness. Another reason for choosing this function is that its value is constant with zero gradient along the vertical edges of the plot, and we can therefore check that termination of the splitting process always occurs at or before the selected level.

We begin by considering what we expect to be the most reliable (but also the most time-consuming) measure, which is a function of data at the vertices of all the triangles within the element (see 5.4 for details); our first task is to select a pair of suitable parameters $\varepsilon$ and $\kappa$, as defined in 5.4. We therefore decide upon a minimum size of grid element, and holding $\kappa$ fixed at this stage we make an initial choice for $\varepsilon$ in the manner descibed in 5.4. However, if the selected value of $\kappa$ turns out to be large in comparison with typical measurements of flatness, there will be a tendency for $\kappa$ to dominate the measurements of flatness with the result that the final grid will tend towards being a regular grid of our chosen minimum size of elements. Conversely, if $\kappa$

is small in comparison with most flatness measurements, splitting of elements will tend to become over-localised, with a high degree of splitting occurring in a few areas of the plot and little or no splitting occurring over most of the plot. Therefore, after inspection of the grid produced with the initial choices of control parameters, it will usually be necessary to adjust $\kappa$ and $\varepsilon$, in such a way that the product $\kappa\varepsilon$ remains approximately constant, until a suitable pair of values is found. While this process of adjustment is being carried out we will not of course wish to carry out any contouring and indeed it is simple to avoid this computationally; also, as we mentioned in Section 2 of this chapter, we can prevent an excessively large data structure from being generated in cases where parameter values have been ill-chosen simply by dimensioning the arrays VALS, IPTR and ITREE so that they are too small to cope with trees of an undesirably great magnitude.

Using the process described above in combination with our measure of flatness the grid illustrated by Figure 6.9 was created. The values of $\kappa$ and $\varepsilon$ used for this plot were 1.0e-03 and 1.0e-02 respectively, values which allow no more than four levels of splitting to occur if we begin, as we do, with a 4 x 4 grid of elements. In fact for this particular value of $\varepsilon$ we will have precisely four levels of splitting for any $\kappa$ in the range [3.81e-06, 3.05e-03]. It is emphasized that if we wished to contour this particular function, which is very well-behaved, using the adaptive method, we would not normally wish to use such a large number of elements as appears in Figure 6.9: the high degree of splitting has been requested simply for illustrative purposes. However contours are plotted in this example in order to indicate their positions in relation to the grid cells.

Figure 6.10 illustrates a comparable grid constructed using our recommended measure of flatness (with the same values of $\kappa$ and $\varepsilon$),

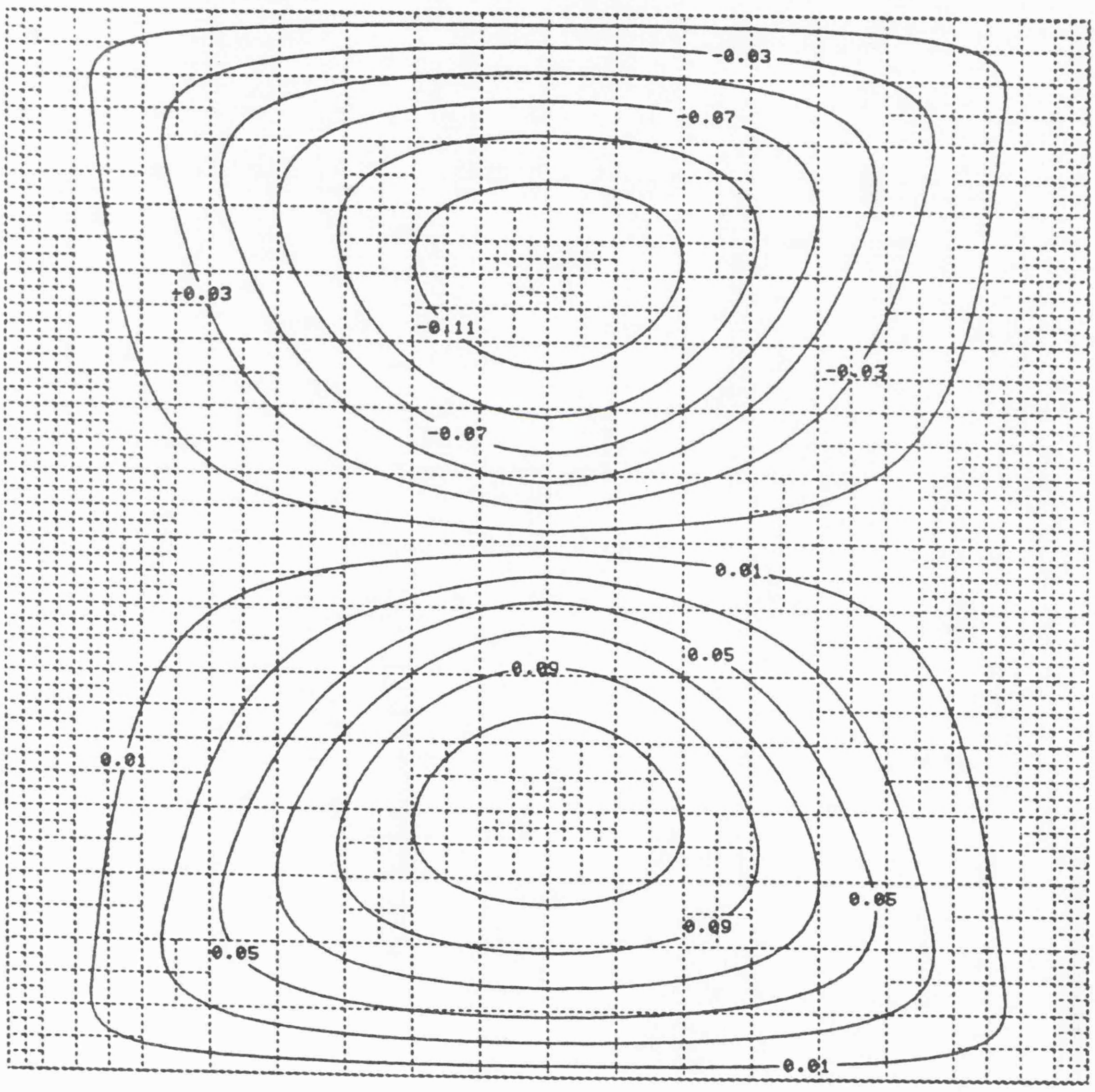


Figure 6.9 Third order error function 3 $c_{112}$ (x, y). Splitting based on 'most reliable' measure of flatness suggested.

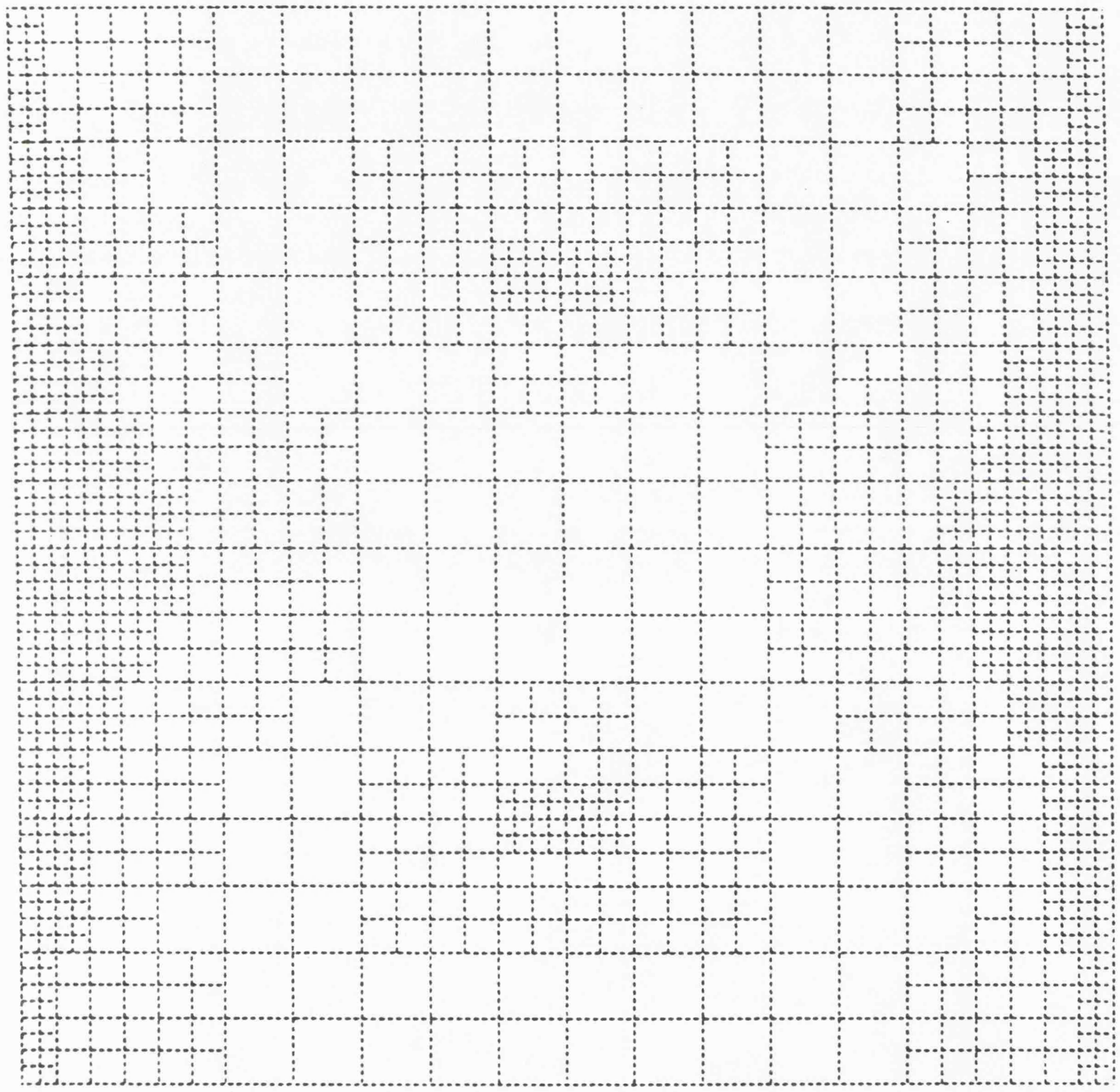


Figure 6.10 Pattern of elements for error function $3\ c_{112}\ (x,\ y)$ when splitting is based on the recommended measure of flatness.

which is a compromise between the method used to produce Figure 6.9 and the crudest suggested method which simply selects the minimum of the five gradients at the vertices and centre of the element. The latter method was also employed as the basis of a splitting criterion to be used on this function, but with a resulting map so similar to Figure 6.10 that it has been omitted; note though that different parameter values had to be chosen in order to produce a similar number of elements to those in Figures 6.9 and 6.10.

Perhaps the most striking feature of these two illustrations is the fact that areas with high concentrations of elements are virtually contour-free, a property which was predicted in the previous chapter. It appears that both measures of flatness have performed successfully in this example; each has succeeded in locating the flattest areas of the plot and a good impression of the overall shape of the surface can be gained simply by looking at the patterns of elements. The differences between the two grids are fairly insubstantial, and the pattern produced by the more time-consuming measure is by no means obviously superior. It is felt that a much more demanding test example might make it easier to differentiate between the two measures.

In fact the computational burden of the more time-consuming measure, which led us to recommend the use of the compromise measure in Chapter 5, turns out to be much less severe than predicted, but is nevertheless greater than that pertaining to the latter measure. We shall therefore not alter our recommendation and we use the 'compromise' measure only in the remainder of this section.

### 6.4.2 Combining vertical error with flatness measurements

We return now to the two test functions which we attempted to contour adaptively on the basis of vertical error in the previous section. Before combining our estimate of the maximum third order partial derivative with a measurement of flatness, we shall examine how the flatness measure performs on its own (i.e. assuming vertical error to be constant at unity) on our first test function. Its performance is illustrated by Figure 6.11, in which the selected values of $\kappa$ and $\varepsilon$ are indicated. We should therefore expect the combination of flatness measure and third derivative estimate to result in some sort of compromise between Figures 6.7 and 6.11.

When combining these two measures the magnitude of the parameter $\kappa$ relative to our gradient measure is no longer simply the determinant of how uniform our grid is:- it now also determines the relative influence of our two measures on the final grid. If $\kappa$ is relatively large then the third derivative measure will have the dominant effect on the final grid, while if $\kappa$ is small the degree of flatness is likely to be more important. Figure 6.12 is an illustration of a parameter choice which it is believed tends to favour the measure of flatness slightly, while in Figure 6.13 the balance appears to be fairly even. Note that the increased magnitude of the product $\kappa\varepsilon$ in Figure 6.13 eliminates all the elements of the smallest size from Figure 6.12. It is clearly going to be extremely difficult to determine the (in some sense) optimum values of our control parameters using this criterion, particularly if this has to be done 'blindly' i.e. with little conception of the appearance of the true contours of the function. Use of this criterion does not therefore appear to be a practicable proposition unless the parameters can be selected automatically in a reliable manner; if such a choice could be made then we might be justified in using this method, though of course the problem remains that we could not hope to measure how good an approximation to the true contour plot we had achieved.

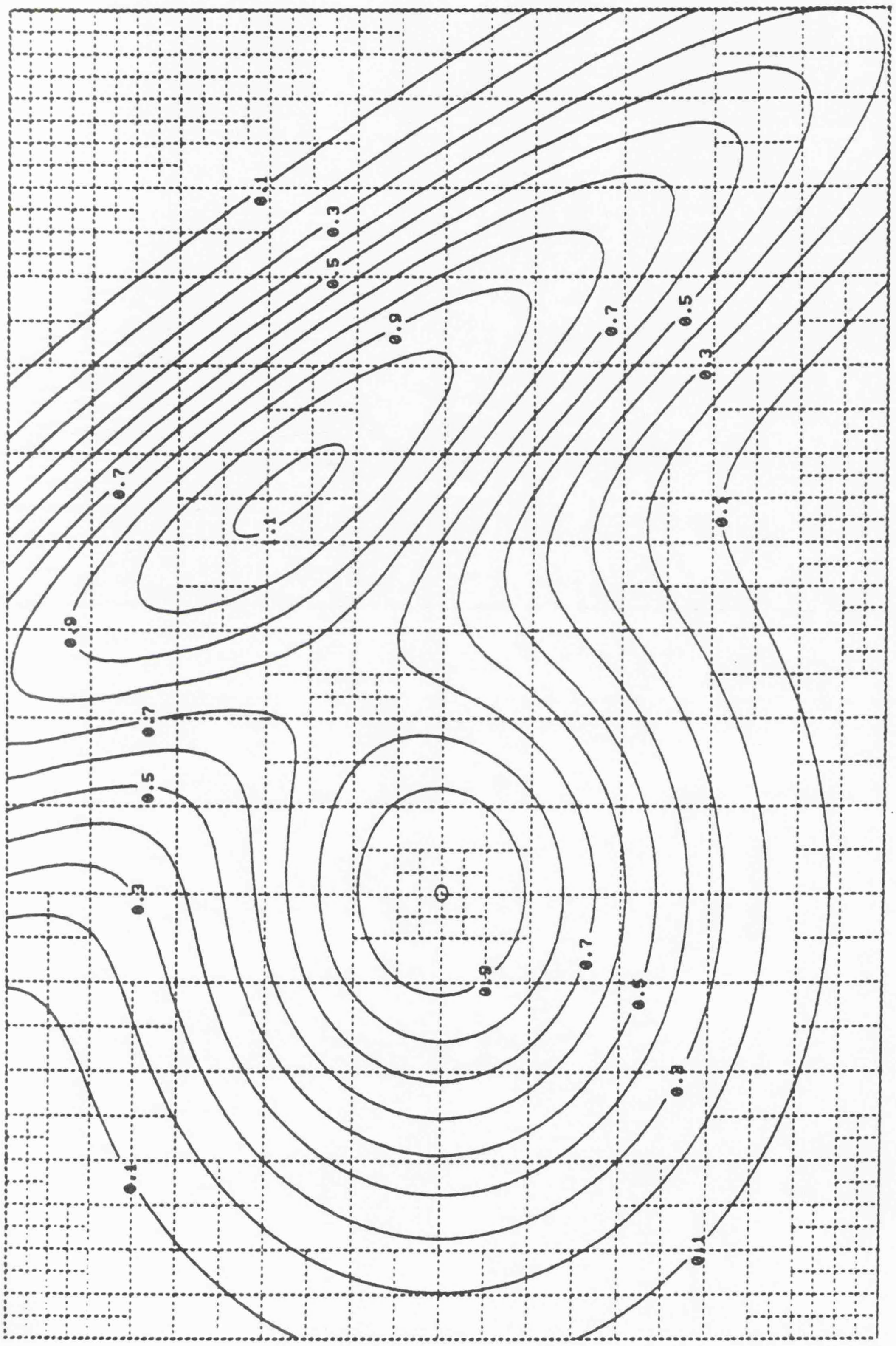


Figure 6.11 Function $f_1$ (x, y). Splitting based on flatness measure alone. $\kappa = 0.01$, $\varepsilon = 0.003$.

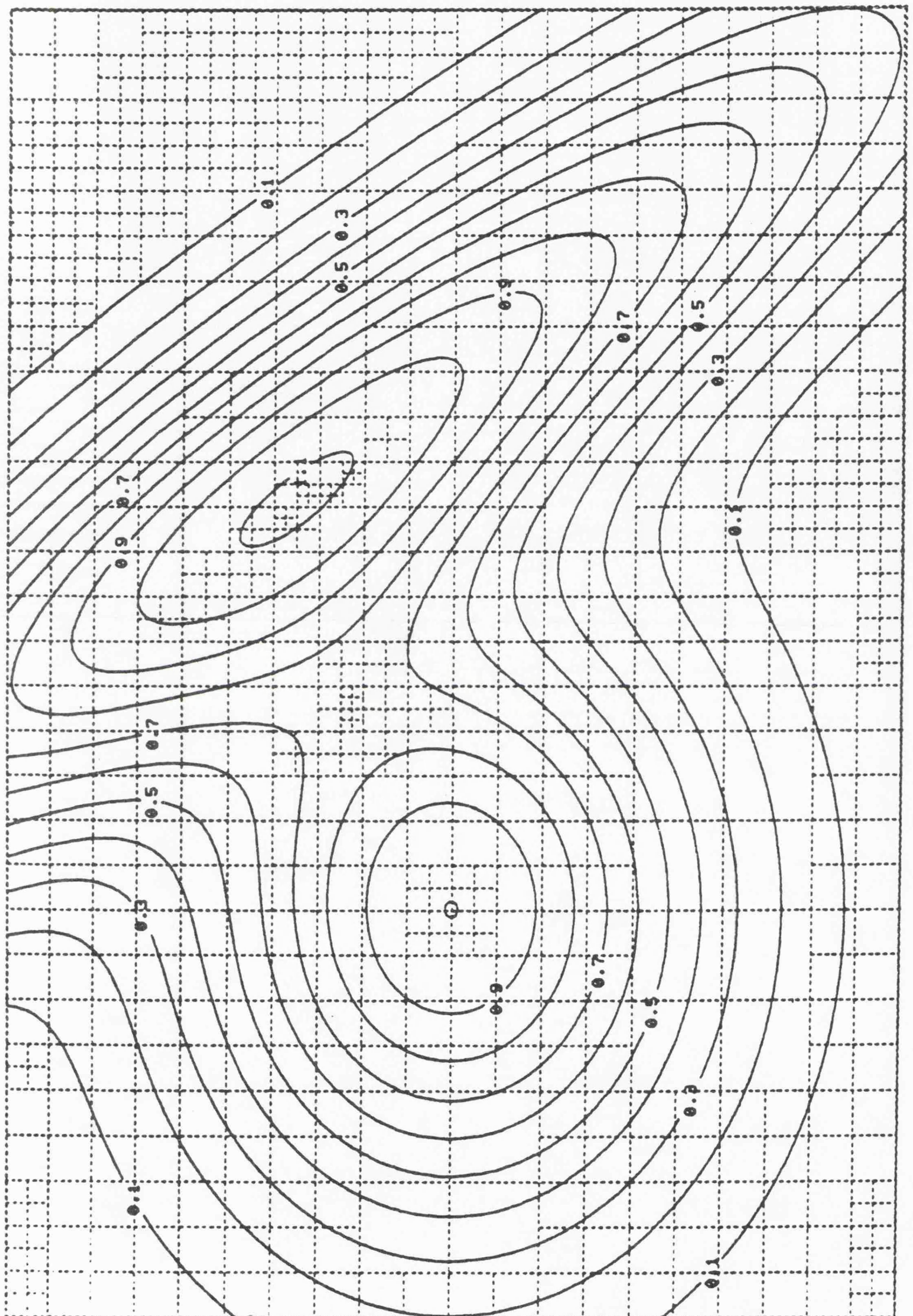


Figure 6.12 Function $f_1(x, y)$. Measure of flatness and estimate of vertical error combined $\kappa = 0.01$, $\varepsilon = 0.002$.

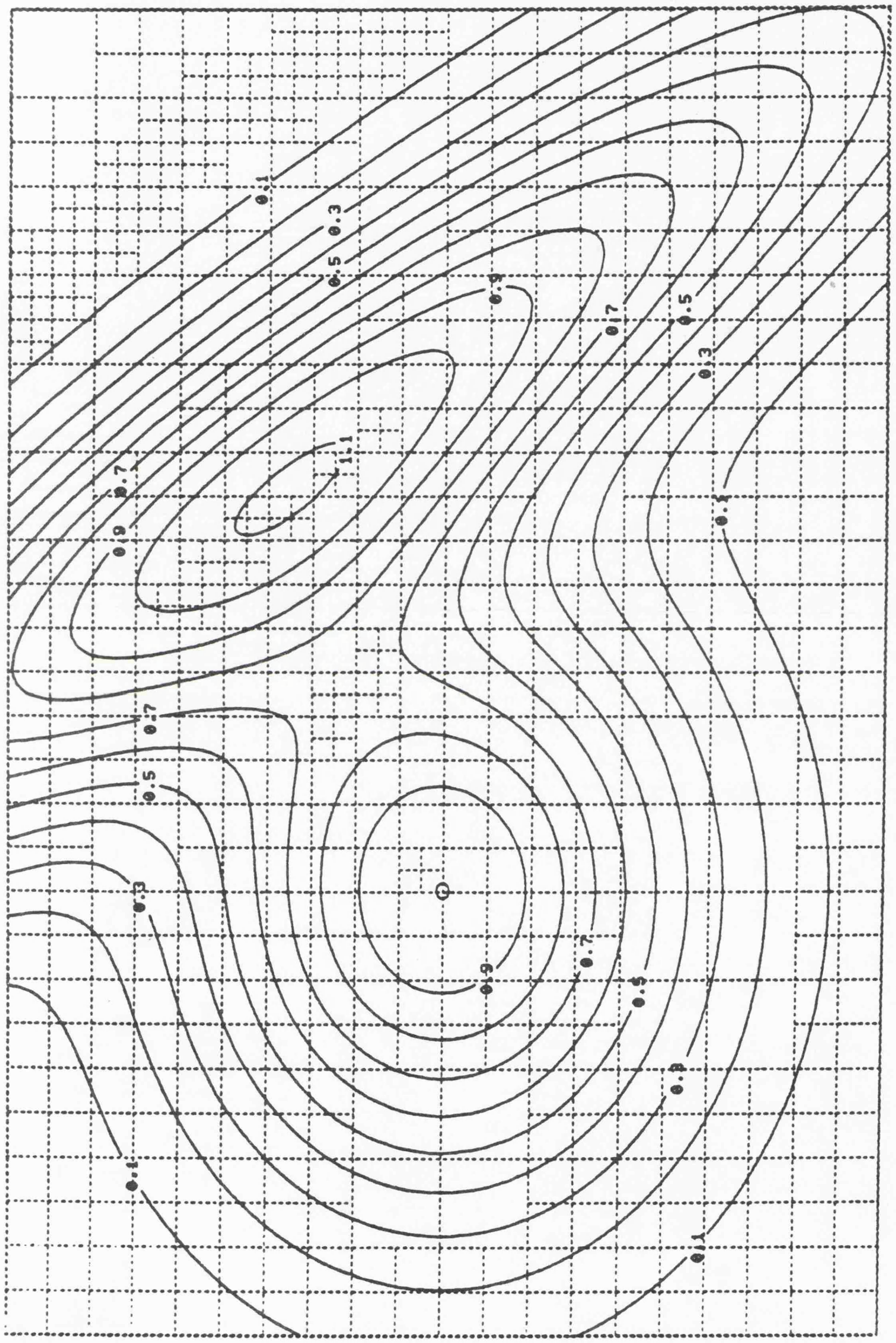


Figure 6.13 Function $f_1$ (x, y). Measure of flatness and estimate of vertical error combined.
$\kappa = 0.05$, $\varepsilon = 0.002$.

We now turn our attention to the second test function and begin once again by examining the behaviour of a splitting criterion based on the measure of flatness alone. After choosing a suitable pair of values for the control parameters in the manner explained above, we arrived at the construction illustrated by Figure 6.14. In this example the flatness measure alone appears to have given us exactly the results we require:- the smaller scale elements are all situated close to the 'peaks' and 'troughs' of the function, while in areas where the majority of contours occur elements tend to be of the larger scale of the two; thus this considerable increase over the number of elements in a 24 x 24 regular grid will normally result in a relatively small increase in the CPU time required to produce a contour, whereas on the few occasions when a contour is situated near to a maximum or minimum of the surface, the method provides the large amount of elements required to produce satisfactory results. Figure 6.15 shows the same contours which were so badly plotted in Figure 6.4 causing no problems on this grid.

Having achieved such satisfactory results using the measure of flatness alone there is little incentive to attempt to combine it with our estimate of maximum error. The only possible improvement that might be hoped for would appear to be a greater degree of splitting near to the origin, where the function is particularly difficult to approximate, and a lessening of the degree of splitting in areas distant from the origin. An attempt was made to achieve such an effect by combining the two measures; however, largely because our third derivative estimate failed to achieve very satisfactory results for the current function, the best results that the author was able to produce were still inferior to those results gained using measurements of slope alone. The best results achieved are illustrated by Figure 6.16.

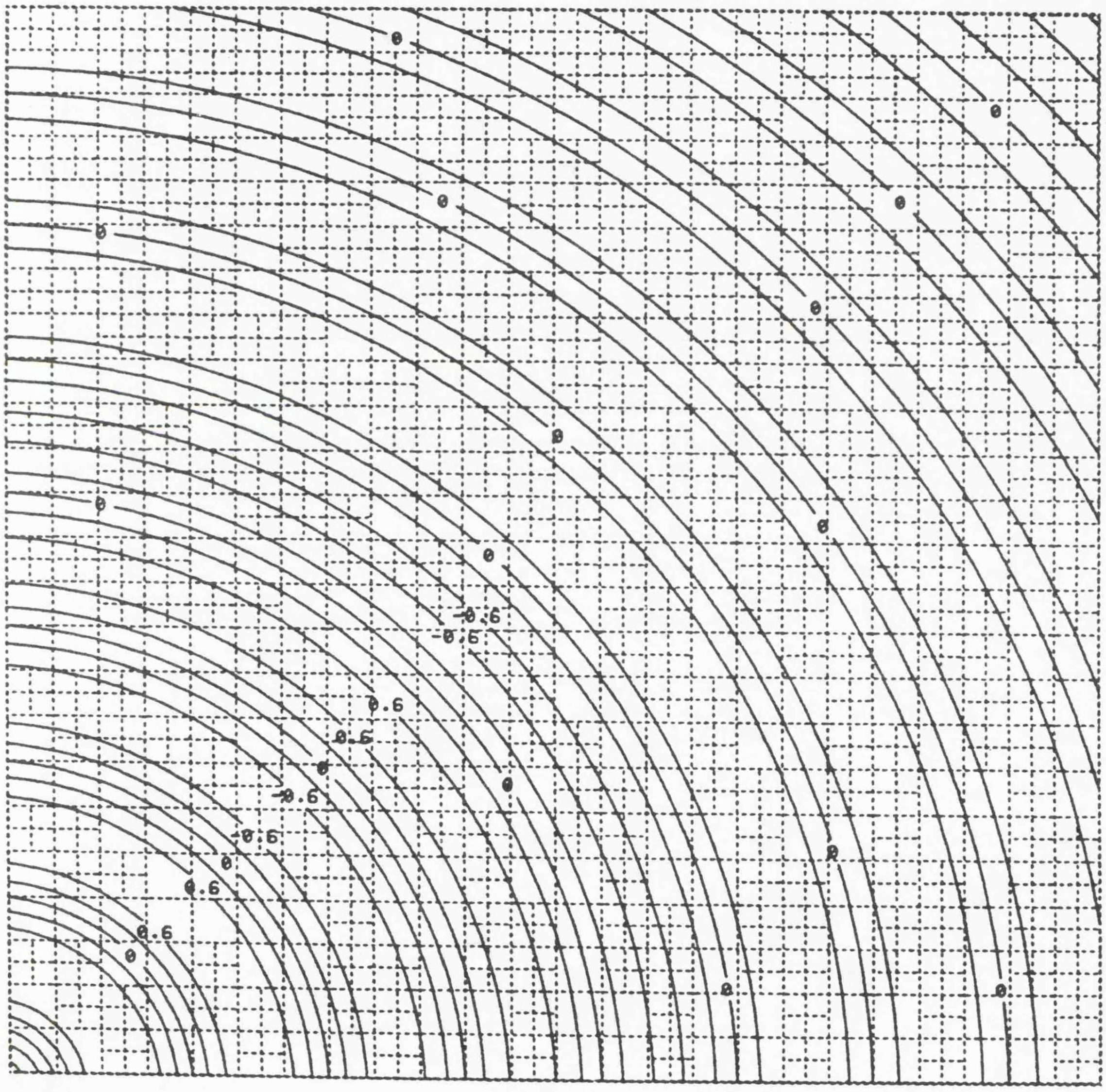


Figure 6.14 Function $f_2$ (x, y). Splitting based on flatness measure alone. $\kappa = 0.05$, $\varepsilon = 0.1$

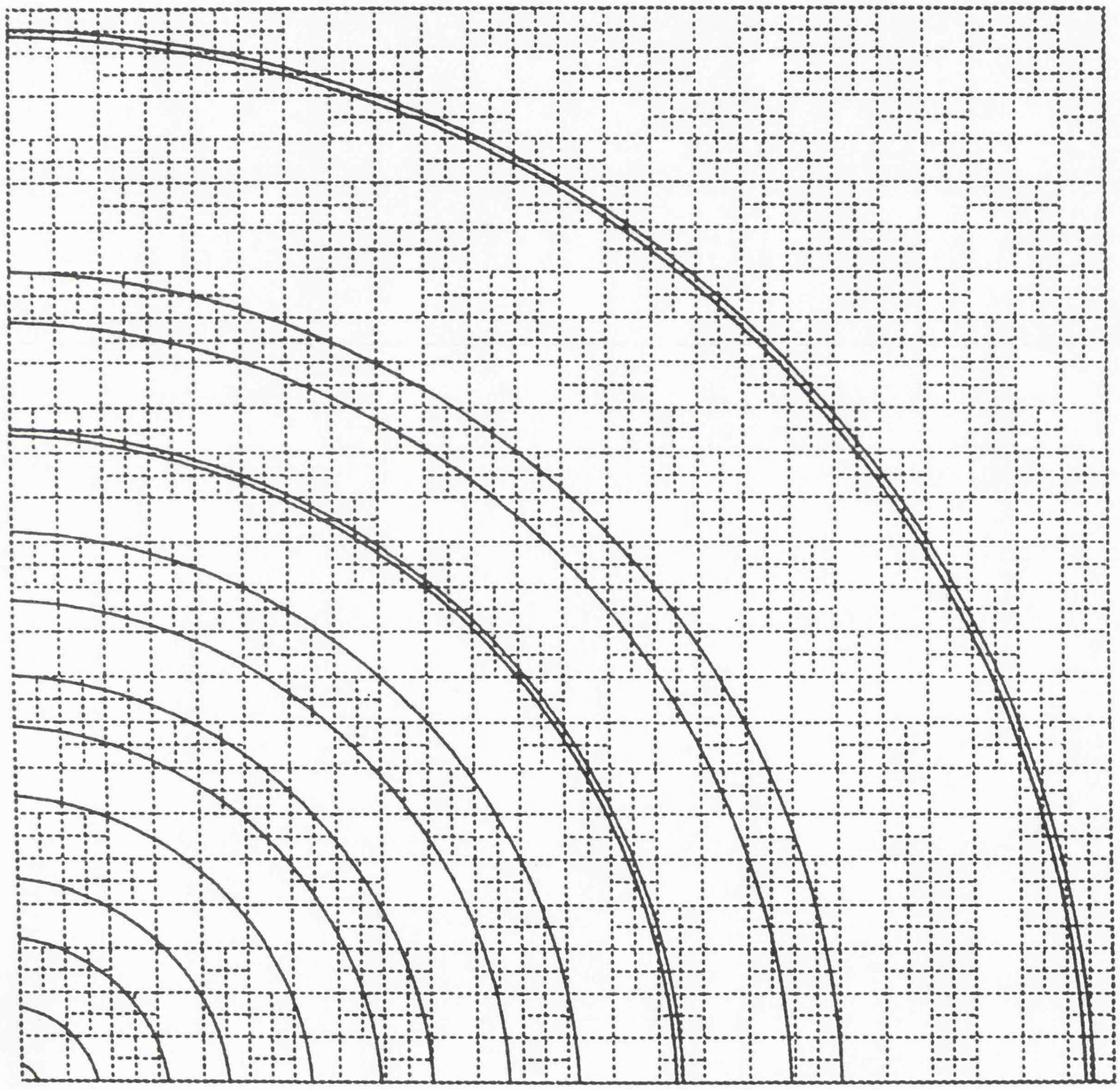


Figure 6.15 Function $f_2$ (x, y). Splitting based on flatness measure alone. $\kappa = 0.05$, $\varepsilon = 0.1$. Same contours as in Figure 6.4.

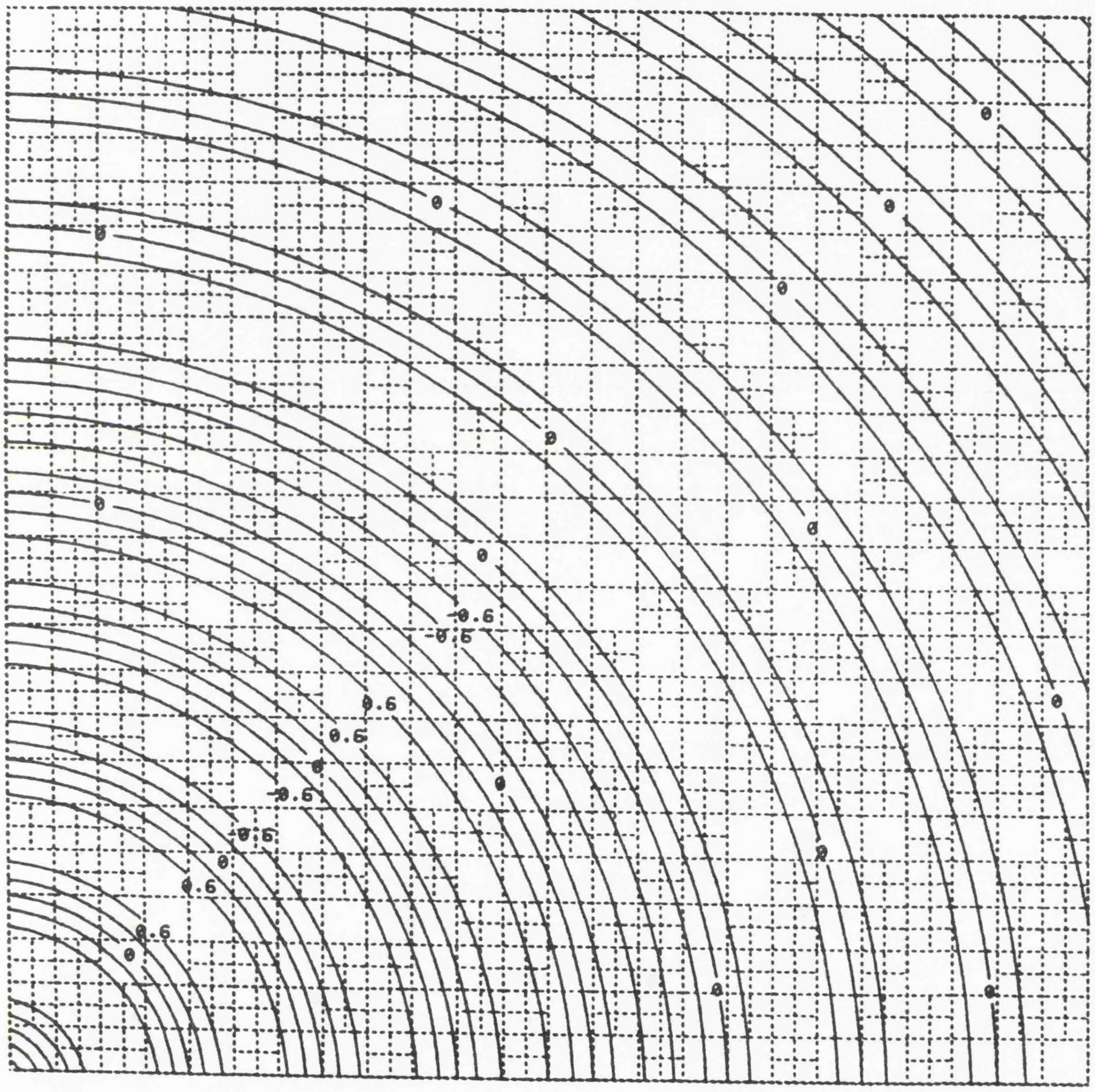


Figure 6.16 Function $f_2$ (x, y). Measure of flatness and estimate of vertical error combined. $\kappa = 0.03$, $\varepsilon = 0.05$.

## 6.5 Discussion

It would be premature to offer any firm recommendations for use of a particular splitting criterion on the basis of the results presented in Sections 3 and 4 of this chapter, partly because these results appear somewhat contradictory. Function 1 seems better suited to adaptive contouring using a criterion based on vertical error only, or possibly vertical error combined with a measure of flatness, while use of a measure of flatness alone appears almost ideal when dealing with Function 2; probably the grid constructed to contour the latter which is illustrated in Figure 6.14 and 6.15 is the most encouraging of all grids presented. However, as we have already noted, choosing the control parameters for either of the methods not based solely on vertical error is such a difficult task that the practicability of these methods must be called into question. Indeed when we combine measures of vertical error and flatness it might seem more appropriate to use three control parameters: at present we are forced to use a pair of parameters to control three variables - the minimum size of elements within the grid, the uniformity of element sizes in the grid, and the relative importance of each of our two measures. Choosing three control parameters would however be an extremely daunting prospect.

The splitting criteria which rely on measurements of flatness or horizontal error are subject to a further problem: throughout sections 3 and 4 we have assumed that we can recognize a 'good' grid when we are presented with one. In practice this is a particularly strong assumption to make and it will be extremely difficult to know what to look for, especially when the approximate form of the contours of the function is unknown. These difficulties are compounded by the fact that it is usually possible to derive an enormously wide range of possible grids. In the examples presented in this chapter we have had the advantage of knowing precisely how the contours ought to appear, and choice of control parameters has still been difficult; these difficulties would be further multiplied in circumstances in which the correct form of the contour plot was unknown.

Choosing the single control parameter in examples which rely on vertical error only presents no great problem. However a defect of the method chosen for estimating third order partial derivatives which has come to light during experimentation makes even the reliability of these methods uncertain. We have shown above how it is possible to keep sub-dividing elements successively in such a way that we may finish up with a pair of neighbouring elements of greatly different sizes (for example the elements labelled A and B in Figure 6.17). Consider the stage of grid construction at which we must decide whether the smaller of such a pair of elements, A, requires subdivision. No matter how many levels it is below its neighbour, B, in the tree, the values and gradients at the two vertices which lie on the edge common to A and B must conform to the values of the piecewise quadratic along that edge. However the

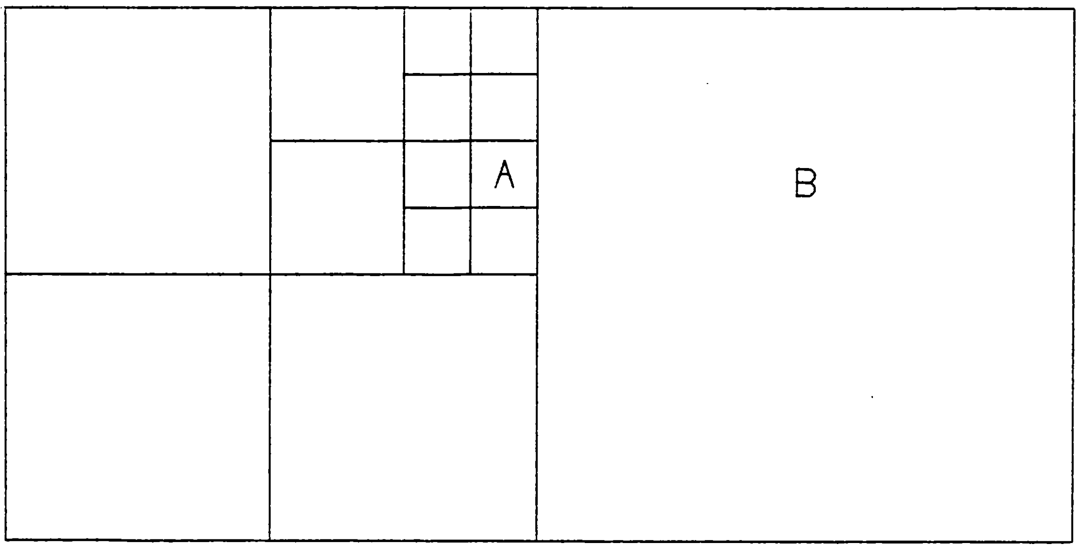


Figure 6.17

values at the other two vertices of element A may not suffer from similar conditions (e.g. as in Figure 6.17) and might be the true heights and gradients of the surface. It is likely because such a large amount of subdivision has occurred in this area that the behaviour of the function here will be relatively complicated. Thus the data at the vertices of element A lying on the boundary of element B may be very different from the true values at these points (and very different from the data at the other two vertices) and may result in such a high estimate of maximum third order partial derivative within element A that it has to be split again. At the next level down we may find that the estimates of third derivative over the new pair of elements bordering element B have risen by a factor greater than 8 and so the splitting process may not terminate.

One potential solution to the problem outlined above is never to allow a pair of elements of more than (say) two levels difference in size to be neighbours. Thus if we were to reach the point during construction of the grid where subdivision of an element which we wished to split would result in such a pair of neighbours occurring we would decide that a mistake had been made in not splitting the larger element. It would therefore be necessary to go back and split the larger element and, if possible, follow through all the repercussions of this on the state of the data structure. Such a policy would however be extremely difficult to implement computationally, and it could also prove to be very expensive to run. A much simpler solution, though not always guaranteed to work, would of course be to choose a larger number of elements in the initial regular grid.

One way of lessening the computational burden of setting up the grid, whatever splitting criterion used, would be to take account of the contour levels requested by the user and never to split an element whose bounds did not contain at least one of the contour levels selected. Such a policy would obviously save an enormous amount of work in those examples discussed in this chapter in which we found the areas where the largest aggregation of elements occurred to be entirely free of contours, and therefore appears to have great potential. There are however one or two inherent problems associated with such a policy; firstly, it may happen that although the bounds for values taken by the piecewise quadratic within an element contain none of the requested contour levels, after some degree of splitting has occurred within that element we might find that some of the selected contours do in fact traverse this area. A slightly more trivial problem is that if such a policy were in operation it would no longer be possible to select contour levels automatically in the way that this is done at present: for we currently use the set of bounds for values of the approximant function within each element to determine the approximate range of contour heights and consequently the contour levels themselves; these bounds are however determined at the same time as the grid is being constructed. Nevertheless in spite of these difficulties the potential savings in efficiency of grid construction to be gained from such a policy make this idea seem very attractive.

It should be noted though that constructing adaptive grids without taking contour levels into consideration does not reduce significantly the efficiency of the contouring process; for use of the bounds (2.6) for the values of the piecewise quadratic within an element means that all elements which we have needlessly constructed will in any case be discarded very quickly.

A further important computational consideration relates to the order in which elements are examined at the contouring stage. In the current implementation elements are simply considered in the order in which they have been generated; however this probably sometimes results in gross inefficiencies in running time because it is likely to cause the typical contour segment produced by subroutine PLTCON to be matched with an unnecessarily large number of unlinked segment ends. If a more systematic ordering of elements could be used it is believed that considerable savings could be made in this respect.

The reader will have noticed that no attempt has been made in this chapter to contour any real examples adaptively, in spite of the fact that locally adaptive choice of grid size seems likely to be particularly useful in such cases, especially when data sites tend to be clustered.

Unfortunately in such cases it is not usually possible simply to supply a subroutine VALUES to evaluate the heights and gradients of our interpolant at any point, since most interpolation methods involve a high proportion of fixed costs, which are often the costs of matrix inversion (or, in the case of Natural Neighbour Interpolation, the setting up of the Dirichlet Tessellation (Green and Sibson, 1978) on which the technique is based) which must be paid before any evaluations of the interpolant (usually relatively cheap) are carried out. Thus to use our locally adaptive contouring method on real examples we must first bind the implementations of the interpolation and contouring stages much more closely together.

However, once it becomes possible to carry out adaptive contouring on real data, a number of other possible splitting criteria can be used. One possibility is to insist on an upper limit (of perhaps just one or two) to the number of data sites lying within each element, since our interpolant, even if not the underlying surface itself, will usually be relatively complex in areas where data sites are clustered. An alternative method might be to ensure that all data sites lie within a specified maximum distance from a node of the grid (and it would probably be necessary in such a case to distinguish between nodes where the true height and gradients had been evaluated and those where an approximation has been made). This is suggested because probably the major defect of using the seamed quadratic element to contour an interpolant is that the height of the approximant does not equal the true height of the surface at the data sites, and the criterion could therefore be used to minimise the effects of this deficiency.

Either of the ideas suggested above might be used as a splitting criterion in its own right, or more likely, could be used as an extra condition in combination with some other splitting criterion.

We have explained why adaptive local choice of grid size is possible when contouring using the seamed quadratic element. However there is no reason why it should not also be used in conjunction with piecewise linear contouring methods, either as a means of improving the smoothness of the contours or to improve goodness of approximation. In the former case a splitting criterion could be based upon changes in direction of contours at the boundary of a pair of cells; and if it were decided that splitting was necessary we might choose to split one or both of these cells.

If we wished to use locally adaptive subdivision in order to improve accuracy of approximation a slightly different approach from any of the ideas suggested for piecewise quadratic contouring would be required; we could not simply rely on the data at the vertices of a cell (whether square or triangular) of the grid to determine whether subdivision was required, as this could not provide us with all the information needed for estimation of second derivatives. We would therefore also have to consider data situated at nearby nodes of the grid.

Such data might also play a useful part in forming a better criterion for splitting using the piecewise quadratic method, indicating the magnitude of changes in second derivatives on crossing the boundary between one element and the next.

In conclusion, it is important to emphasise that the concept of an adaptively chosen grid for contouring is very much in its infancy and we have only been able to present a rather speculative introduction to the subject. A wide variety of possible criteria for local sub-division of squares has been suggested; although those which we have tested appear to have shortcomings which might prevent them from being used widely, a number of promising possibilities still remain. This is an area of study which will surely reward further work.

# CHAPTER 7

# A BRIEF COMPARISON OF 14 CONTOURING PACKAGES

## 7.1 Introduction

In this final chapter we shall describe and compare the features and algorithms contained in a wide variety of the contouring packages currently in use both in industry and in scientific research. The comparison is carried out solely on the basis of user documentation or, in some cases, sales literature, which is of course insufficient information for a full evaluation of the relative merits of the packages. Ideally we might like to compare the performances of the packages in contouring a number of standard data sets, but such a mammoth undertaking is practically impossible within the context of this thesis. Equally, we would like to be able to assess in detail the mathematical ideas on which each package is based: such information is often unavailable or offered only in the most general of terms, with numerical details simply being buried in the computer code. In any case no comparison is likely to be sufficient to isolate a 'best' package from the fourteen, for the priorities and needs of all potential users are different. Later in this section we identify and discuss six important criteria which the potential buyer of a contouring package must take into consideration before deciding on his choice.

Although the comparison cannot be expected to produce a 'recommended best buy' it does at least give an impression of the considerable diversity which exists among the packages that we present; the one important exception here is in the contouring methods used, which are all, or almost all, of the piecewise linear type. The selection is by no means exhaustive but it provides a large and, it is hoped, representative sample from the available spectrum (and also includes some packages which are not generally available, but have interesting features justifying their inclusion in this survey). For convenience the packages have been grouped into four categories:- 'mainstream' packages, which are generally available for purchase and were apparently written primarily with a view to commercial exploitation; a group of packages written largely for the oil exploration industry, and therefore incorporating a number of specialised features which in all cases include the ability to accommodate geological faults; some packages written for operation at a single organisation or installation; and finally a pair of packages which specialise in a subset of the contouring process.

One body of contouring packages is not considered at all in this chapter: that is packages designed exclusively for use in combination with raster terminals and plotters. Plotting contours on raster devices is a separate and quite different problem from that of contouring on vector plotting devices, and it is the latter problem to which we have tacitly restricted attention in this thesis. The vector problem can be thought of as a more general problem than the raster one, not simply because contouring on vector devices is currently more prevalent than contouring on raster devices (a situation which cannot be guaranteed to continue indefinitely), but because it is considerably easier to convert vector information to raster information than the converse. In addition by its very nature contouring is inherently a vector rather than a raster problem.

Before discussing the fourteen packages in turn we shall consider the important properties that one should be looking for in a contouring package. This thesis has focussed attention heavily, and it is believed rightly, on matters relating specifically to the contouring algorithm or method employed (accuracy and visual smoothness of contours, as well as mathematical understanding of the method in use), but this is only one of six criteria of considerable importance which the author has succeeded in identifying. We discuss the other five below, paying particular attention to the performance of the CONICON package.

(1) Features offered by the package

This area received a fairly substantial amount of attention in Chapters 3 and 4, and is the one about which we can derive most information from user documentation and sales literature. Undoubtedly the most important feature, since it is an essential part of any contouring package, is contour annotation. At first sight there might appear to be little to say about this, but this is far from the truth. In Chapter 4, we discussed the alternative strategies which may be used in an attempt to make all labels in a plot easily readable:- it seems universally agreed that labels should not be placed in areas of high curvature, but packages differ in their treatment of areas where slope is great and contours are closely bunched. Of those packages which attempt to tackle this problem some try to avoid placing labels in such areas altogether, while others take elaborate steps to ensure that no contour crosses any label wherever it occurs. In Chapter 4 we argued that the former course is to be preferred, because it tends to be much cheaper and because labels occurring in areas of high gradient can be difficult to 'place' on the correct contour anyway. We also considered the alternative ways of orienting labels - which of these is preferable seems to depend upon the complexity of the contour plot:- as long as they are not obscured the placing of upright labels throughout is recommended, but when bunching of contours is widespread it seems preferable for labels to follow contour orientation. This is also desirable when much other information is being plotted, as in topographical mapping.

We shall see that some packages label contours only with small integer values and use a key to identify the precise level of each contour; the most primitive packages (or primitive options within packages) plot contours in short linear sections and can therefore label contours only where they meet the plot boundary. Other points of interest when we consider annotation policy are the degree of control which the user exercises over label size and format, over minimum inter-label distance on a single contour, and over which contours are labelled and which are not.

Crosshatching - strictly speaking with a key provided - is probably the one feature which might be considered a substitute for (and in some cases an improvement upon) contour annotation. However not one of the fourteen packages examined in this chapter offers such a facility. In the author's knowledge CONICON is unique among vector-oriented packages in this respect.

Further graphical features of importance to many users are the ability to contour non-rectangular areas, contour thinning in areas of high density, the marking of local maxima and minima with appropriate labels, placing 'hachures' or tick-marks on the downward slopes of contours and the ability to use several line styles. At present CONICON offers neither contour thinning nor the opportunity to plot hachures; however as we have explained above thinning could be put into practice simply by controlling the plotting of contours according to surface slope, and a facility for plotting hachures could also be added with little difficulty (conic sections are always generated with high ground on the right). There is also scope to refine the contour suppression feature, perhaps to contour the area within an arbitrary polygon, though this would undoubtedly be a more difficult problem to solve. Linestyles are a device-dependent feature, unless generated by software (a relatively expensive alternative).

Many contouring packages also offer non-graphical features such as volumetric and areal calculations, though such features are arguably more closely related to interpolation than contouring; thus if the CONICON package were to include a facility for volumetric calculations it might be more sensible to use the cubic element introduced in Chapter 2 as the basis for such calculations, rather than the quadratic element, for reasons of accuracy. However if we wished to calculate the volume above or below a particular contour level such an approach would be liable to lead to (usually very small) numerical inconsistencies.

Finally, we must mention application-dependent features, and in particular 'faulting' facilities. In the specialised area of oil exploration such facilities have almost become a prerequisite for any contouring package. A fault is of course a discontinuity in a surface and therefore, given the assumption of $C^1$ continuity of the underlying surface, faulting has not been incorporated into the CONICON package. Indeed the extension of the package to include such a facility would be a considerable undertaking, involving much effort both on the theoretical and computational sides. We make no attempt to solve this problem in this thesis.

(2) Efficiency

Efficiency, in common with some of the other factors which we identify in this section, cannot be properly assessed simply from the examination and comparison of user documentation. When we discuss the efficiency of a package, conventionally we break this down into two constituents, its CPU usage and its memory needs. While the latter may be gauged approximately from the information available in user manuals, the former can only be measured properly by benchmarking. Nowadays, with machines possessing virtual memories becoming the rule rather than the exception, CPU usage will usually be regarded as the more important of the two and in some cases its importance may be critical.

Although it is not possible to provide many 'hard figures' to support our arguments, there are good reasons for believing that the CONICON package will not compare unfavourably with other packages with regard to efficiency. The package will of course appear slower than packages employing piecewise linear algorithms if comparisons are made on the basis of a fixed grid size (for example, when comparisons were made with the ECMWF package in Chapter 4, a ratio of approximately three to one was reported); however CONICON has been shown to be capable of producing high quality contours using a small fraction of the data required by piecewise linear methods, and it is in the light of this fact that comparisons should be carried out and judged. As we have pointed out before, this property of the package may sometimes result in important CPU savings in the area of data value generation, beside which times used by contouring algorithms themselves will in some cases be insignificant.

The ability of CONICON to get by using a coarse grid is particularly important from the point of view of memory (if this is in short supply), for it means that the necessity to store five floating point values for each point on the grid, (one height, two gradients and a pair of bounds) in addition to the workspace required for contour linkage, is unlikely to be a handicap in dealing with all but the most elaborate data sets. In cases where memory is a particularly precious commodity CONICON memory usage can be (and indeed has been) improved. Firstly, if gradient values are unavailable and have to be estimated they need not be stored but may be calculated cheaply as and when required (this has been shown to have a slightly beneficial effect on CPU usage at the ECMWF installation, where gradient values are generally unavailable - presumably because it has eliminated some expensive addressing); and secondly the values in the array ZLIM can be packed into, say, 6 bit integers (this would assume a maximum of 63 contour levels), in such a way as to indicate the pair of contour levels between which a particular bound lies, with no loss of effectiveness and a considerable reduction in use of memory. However on most mainframe computers the amount of memory typically used by CONICON is not likely to approach the limit of availability and efforts of this nature are therefore unnecessary.

(3) Financial considerations

In the real world it is not always possible to obtain the package which best suits an installation's needs, simply because it is too expensive. This thesis is probably not the best place to discuss prices of contouring packages (and indeed no attempt has been made to obtain the relevant information), but the importance of financial considerations should not be forgotten. Such considerations do not apply solely to the purchasing of a package, but are also of relevance when an organization considers the amount of time which will be involved in training personnel to use the package. The importance of the latter will depend on the nature of the organization purchasing the package and the experience of those individuals who will be required to use it. This brings us directly to the next factor.

(4) Ease of use: the user interface

The user interface of the CONICON package is a relatively low-level one and requires the user to have at least an elementary understanding of programming in Fortran. A complete contour plot can be set up with a single subroutine call, but because CONICON has no concept of a default linestyle, labelling policy, etc, the user is required to set the values of a relatively large number of parameters. Since the package was designed primarily for use in an academic or scientific environment it is believed that users will have little difficulty in learning how to use the package:- indeed once the user has created his first contour plot subsequent ones are very unlikely to cause any problems.

In the packages which we examine in this chapter a number of different approaches have been taken to user interface design, though all or most packages have been written in the same language, Fortran. Many packages define a set of default parameters specifying linestyles, number of contours, label positioning and format, etc. which are normally stored in labelled Common. Users wishing to alter the values of these parameters may do so by calling an individual routine (normally with just one or two arguments) for each parameter, while the user who requires simply to examine quickly the nature of a surface may do so by specifying a matrix of surface heights and possibly the values of one or two parameters, before making a single subroutine call. In this way such packages are deemed to be very easy to use while retaining a wide range of facilities under user control. There is of course no reason why a user interface of this type could not be built on top of CONICON - with the one caveat that CONICON contains a relatively large number of arrays for workspace, which might usefully be reorganised before such an addition were to be implemented.

Other packages require the user to issue a sequence of commands to set up values of parameters, read data etc. and finally construct the plot. Users are not required to have any knowledge of programming in any language, but it is felt that users who do have some Fortran programming experience would need a relatively large amount of training in the use of such packages; on the other hand the user possessing no knowledge of Fortran, though he might experience more difficulty in learning to use a package based on Fortran subroutine calls, would at least in doing so be gaining a skill which might prove to be of some benefit in other areas of work. Packages with user interfaces based on subroutine calls are therefore preferred by the author to those with command-based user interfaces.

Finally, a few packages provide an interactive user interface, some even going as far as to allow the user to alter contours interactively after they have been plotted. Insofar as such user interfaces simplify the use of a package they are to be recommended; however facilities which allow a user to edit a contour plot interactively must be handled with extreme caution.

(5) Reliability and degree of customer support

One important factor about which little information can be derived from user documentation or sales literature is the reliability of a package. This will be particularly difficult to assess if it is intended to install the software on a machine on which it has not previously been run. Closely related is the level of supported provided - the importance of this varying in inverse proportion with reliability.

In the sections which follow we provide descriptions of fourteen contouring packages which for ease of assimilation and fast reference have been broken down under a number of headings. It should not be forgotten however that in most cases we are unable to provide any indication of the efficiency, cost or reliability of a package - it is therefore recommended that judgement of the packages should not be based solely on the information presented below.

## 7.2 <u>General Purpose or 'Mainstream' Contouring Packages</u>

GPCP      Surface II Graphics System      NAG Graphical Supplement

GINOSURF (Mark 1)      'Surrender'

The packages which we examine in this section are all, like CONICON, general purpose packages which, it is believed, were not written specifically for any one particular application area or installation.

(1) <u>GPCP-II: A General Purpose Contouring Program</u> and <u>Supplement to GPCP-II</u>

<u>Calcomp Applications Software (1972, 1974)</u>

TYPE OF PACKAGE

Carries out both interpolation and contouring. No other means of surface display is provided.

## INTERPOLATION METHOD PROVIDED

A 'projected slope' method:- Gradients are estimated at the control points by a weighted least squares planar fit and interpolated values are then calculated as a weighted average of the projections of planes fitted at 'neighbouring' points. The user controls the number of neighbours (defined by distance alone) used in such calculations and the weighting factor ensures smoothness of the interpolant.

## CONTOURING ALGORITHM USED

A refinement of the piecewise linear method. After interpolating, if necessary, to values on a rectangular matrix, each cell is divided further into a subgrid. A bicubic polynomial is fitted across each cell of the larger grid, with smoothness preserved across cell boundaries. This function is evaluated as and when needed at the nodes of the subgrid and contours are traced across subgrid cells using inverse linear interpolation. The sixteen parameters of the bicubic function are fitted from four values at each corner of the cell - the interpolated height plus estimates, based on other grid values, of the first order partial derivatives and the second order mixed derivative. The method results in relatively smooth contours which do not cross (see Figure 4.19).

## LABELLING POLICY

Labels follow contour orientation, each digit being oriented individually. Labels often appear 'upside down'.

OTHER GRAPHICAL FEATURES

* Thinning - in areas where contours are bunched contours of the standard linestyle are omitted while bold lines remain.
* Hachures - the user controls their size and location e.g. they may be restricted to appear only on bold lines, in closed loops containing a local minimum.
* Contour suppression within an arbitrary polygon - the definition also allows the creation of 'holes'.
* Production of stereoscopic pairs for viewing through tinted lenses - an unusual feature.
* [Supplement]. Plotting of vertical cross sections.

NON-GRAPHICAL FEATURES

* Complete user control over contour levels.
* [Supplement]. A comprehensive collection of areal and volumetric calculations.
* Contouring over skewed and rotated grids.
* [Supplement]. Least squares trend surface and residual plotting (up to 10th order).

USER INTERFACE

Of the 'sequence of commands' type. Various defaults provided.

COMMENTS

One of the better packages from the point of view of contour quality. A good contour suppression feature and a fairly wide range of facilities. The interpolation method is suspect though - see Section 4.5.

(2) Surface II Graphics System    Sampson (1975, revised 1978)

Kansas Geological Survey

TYPE OF PACKAGE

A general package for the display and analysis of 2D surfaces. Includes interpolation, a crude form of smoothing, contouring and perspective block diagrams.

INTERPOLATION METHOD PROVIDED

(i) A method very similar to that used in GPCP-II, but with a selection of weighting factors available. In addition several alternative methods of defining 'neighbours' are provided:- the user controls the number of neighbours, which may be located by a standard nearest neighbour search, by a quadrant search (a minimum number of neighbours must be located in each quadrant - obviously rotation dependent) or by an octant search. Alternatively all points lying within a fixed radius r of the point of interest may be deemed to be neighbours.

(ii) Universal Kriging.

SMOOTHING METHOD

Following interpolation, the value at each grid point is replaced by the mean of all values within a user-defined radius.

CONTOURING ALGORITHM USED

The most basic piecewise linear method, with no subdivision of grid cells (see Figure 4.21). Smoothing of contours is available (with of course the consequent risks of contours crossing) by fitting circular arcs between the points defined by the piecewise linear method.

## LABELLING POLICY

Labels follow contour orientation. The package attempts to avoid labelling in areas of high curvature, but takes no account of gradient or contour density when placing labels. The user has a large degree of control over inter-label distances, label size and label format.

## OTHER GRAPHICAL FEATURES

* Thinning - user-specified minimum inter-contour distance.
* Hachures - with user control over size and location.
* Contour suppression - the user specifies those cells in which this takes place by setting grid values large and negative.

## NON-GRAPHICAL FEATURES

* Complete user control over contour levels.
* Least squares trend surface analysis - only 2nd order.
* Contour plots of distance to nth nearest data site - for assessing accuracy of standard contours.
* 'Filtering' - a matrix multiplication of a point and its neighbours is carried out to provide a weighted spatial moving average.
* Contour plots of derivative in any specified direction - derivates are estimated at the grid points and then contoured in the usual way.
* All grid values outside a specified range can be amended to lie on the edge of that range.
* 'Error Analysis' feature - calculates the difference between the true surface heights and the piecewise linear surface at the data sites, then interpolates and contours the resulting 'error surface'.

USER INTERFACE

As in GPCP-II, a sequence of commands is provided by the user to control program flow. Default parameter values may be assumed.

COMMENTS

The wide variety of features offered by this package cannot disguise the inherently poor quality of the contouring method. Many of the features are in any case rather unsophisticated and could trivially be replicated in other packages.

The multitude of possibilities for interpolation, all of which lead to different results, is likely to prove bewildering to the inexperienced user. In the author's opinion a single interpolation method with good mathematical properties which defines neighbours uniquely and in a natural way would be preferable.

## (3) NAG Graphical Supplement

Numerical Algorithms Group (1981)

TYPE OF PACKAGE

A general graphics package which includes facilities for contouring (and interpolation).

CONTOURING ALGORITHMS USED

Two piecewise linear algorithms due to Heap & Pink (1969). The first plots straight lines across each grid cell, ambiguities being resolved using the rule: high ground on the right. The second subdivides grid cells into four triangles, taking the mean of the four corner values as the centre height.

Two methods for contour smoothing are provided:

(i) Butland's method (1980). Fits smooth but tight-fitting curves, with reduced risk of contours crossing (see Figure 4.17).

(ii) McConalogue's method (1970, 1971). Fits smooth, free-flowing curves. Rotation dependent.

## LABELLING POLICY

Contours are labelled with upright integer values and a key is provided for translation. The package attempts to avoid labelling where there is 'insufficient room'. The user specifies label size and the interval at which contours are labelled.

## OTHER GRAPHICAL FEATURES

None.

## OTHER FEATURES

* Full user control of contour levels - or automatic selection.

## USER INTERFACE

The user calls a single Fortran subroutine with a relatively long argument list. No concept of default parameters.

## COMMENTS

Although first released in 1981, this package seems little or no better than many packages written ten years earlier:- it offers few facilities and employs unsophisticated contouring algorithms.

(4) Ginosurf (Mark 1)

CADC, Cambridge

TYPE OF PACKAGE

A general package for displaying 2D surfaces, with interpolation as well as contouring and plotting of isometric projections.

INTERPOLATION METHOD PROVIDED

A method due to Falconer (1971). Carries out a local weighted least squares fit of a paraboloid (4-parameter) surface. Can be demonstrated to produce surfaces with discontinuities in first derivative in certain special cases.

CONTOURING ALGORITHM USED

A piecewise linear algorithm due to Heap (1974) which divides grid cells into four triangles.

Smoothing of contours is possible using the method of McConalogue (1970, 71).

LABELLING POLICY

Labels follow contour orientation, and minimum inter-label distances are under user control.

OTHER GRAPHICAL OR NON-GRAPHICAL FEATURES

None.

USER INTERFACE

Plots are created by a Fortran subroutine call. Default parameter values may be overriddenby making extra subroutine calls prior to plot construction.

COMMENTS

* Contours in a single plot can only be plotted at regular intervals (though other contours may be added by overplotting).

* The following additional features are promised in Ginosurf Mark 2:- volume and surface integrals, contour suppression and plotting of cross sections.

* The fact that a defective interpolation method is provided is disturbing:- though discontinuities in first derivative are unlikely to occur in practice there is no reason to believe that such irregularities will not sometimes be closely approximated.

* We are told that run times vary in proportion with the number of grid cells (cf CONICON, where run time is approximately proportional to the square root of the number of cells).

* It is to be hoped that the coding in the package has been written with considerably more care than the documentation, which contains one contour plot ('Test Example 2') which demonstrably does not represent the data set which it purports to represent. A similar plot has been simulated using the CONICON package by inserting an extra value of zero at the start of the data set and thereafter reading all values one step away from their true positions.

## (5) 'Surrender' - A Subroutine Package for rendering bivariate surfaces

Computing Centre at the University of Trandheim

Zachrisen (1979)

TYPE OF PACKAGE

For producing contour plots and perspective block diagrams from rectangular matrices of heights.

CONTOURING ALGORITHM USED

The simplest form of piecewise linear contouring, with no subdivision of rectangular grid cells. Contour smoothing is provided using spline interpolation.

LABELLING POLICY

Labels follow contour orientation and label format and size is under user control.

OTHER FEATURES

None

USER INTERFACE

The package has no concept of default parameter values, but as a result of the paucity of features available the single subroutine call which is required has a relatively small number of arguments.

COMMENTS

A crude package, with very few facilities and at least two defects:- contours within a single plot may only be placed at regular intervals, and run times appear to increase linearly with the number of grid cells.

## 7.3 Packages for the oil exploration industry

CPS1 ZMAP SDL (Surface Display Library) MCS (Mapping-Contouring System)

The packages described in this section have all apparently been designed and marketed with the objective of sales to the wealthy oil exploration industry very much to the fore. In addition to all or most of the features which appeared in the previous section, these packages all incorporate facilities which allow for faulting (i.e. discontinuities) in a surface when plotting contours.

This group of packages is undoubtedly the most sophisticated group described in this chapter. However it has unfortunately only been possible to make evaluations on the basis of sales and advertising literature rather than user documentation and as a result explicit details of the contouring algorithms employed are not given in general. In addition details of the user interface and the degree of user control over parameters are lacking.

It must also be borne in mind that the packages described in this section are expensive: for example, in the case of the SDL and associated packages, an installation outside North America would have to pay a sum in the region of $75,000 for a fully supported package incorporating interpolation, three methods of surface display and a faulting capability. There is little reason to suppose that the other packages will be very much cheaper.

## (6) CPS1: a contour plotting system

Radian Corporation, Austin, Texas

### TYPE OF PACKAGE

A general package for the creation and display of 2D surfaces. It incorporates interpolation, smoothing and three means of surface display:- contouring, perspective block diagrams and projected contour displays (or 'floating contour projections').

## INTERPOLATION METHOD USED

The interpolation techniques used create surfaces with 'smooth minimum curvature between data points'. No further details known.

## SMOOTHING METHOD USED

Not known.

## CONTOURING ALGORITHM PROVIDED

Uses a regular grid with subgridding, probably very similar to the method employed in GPCP-II.

## LABELLING POLICY

Labels follow the orientation of contours, each digit being oriented individually. The user has 'complete control' over labelling and labels automatically avoid areas of high curvature.

## OTHER GRAPHICAL FEATURES

* A sophisticated facility for the creation of cross-sectional views, which accommodates non-vertical faults.
* Hachures.
* Thinning.
* Contour suppression within/outside arbitrary polygonal areas.

## NON-GRAPHICAL FEATURES

* Filtering.
* Volumetric and areal calculations - with or without faulting.
* Trend surface analysis - up to 8th degree.
* 'Complete user control' over choice of contour levels.
* Several others, including an annual users' conference.

SPECIALISED FEATURES

* Faulting: contours are drawn right up to the fault trace. The package defines and accommodates two types of faulting:- an 'opaque' fault treats data on opposite sides independently, while a 'translucent' fault 'uses regional characteristics across the fault while preserving the local discontinuity'.

USER INTERFACE

The user issues a sequence of commands via control cards which set up parameter values, read data and create the plot.

COMMENTS

Although the available evidence is rather insubstantial, the variety and capabilities of the features offered by this package nevertheless appear very impressive. Contour curvature is however imperfect and one or two other possible defects emerge from the sales literature: for example, the thinning feature is aesthetically unattractive, perhaps because the algorithm adheres too strictly to a 'minimum distance' rule.

## (7) <u>ZMAP</u>

<u>Zycor, Inc. Austin, Texas</u>

TYPE OF PACKAGE

A general surface display package, including interpolation, smoothing, contouring and plotting of perspective block diagrams. The package provides an interactive user interface with 'powerful editing tools'.

INTERPOLATION METHOD USED

Several methods are available, including 'moving weighted least squares', 'moving weighted average', 'closest point (polygon) method', projected slope method. In each case the user controls the number of neighbours used in interpolation calculations.

SMOOTHING METHOD USED

Not known.

CONTOURING ALGORITHM USED

The algorithm takes as input data surface heights on a rectangular grid. The sales literature claims that contouring is 'built around a unique new algorithm that significantly reduces execution time and the output load placed on the plotting equipment ... contour point spacing automatically decreases in rough areas and increases in smooth areas to minimise the number of points required to satisfactorily define the curves'. Thus the method is probably non-linear and possibly piecewise quadratic.

A second, coarser contouring method (presumably piecewise linear) is also implemented.

LABELLING POLICY

Labels follow contour orientation. The user controls 'labelling rate, size and accuracy'.

OTHER GRAPHICAL FEATURES

* Thinning.
* Hachures.
* Contour suppression inside or outside an arbitrary polygon.
* Plotting of cross sections (optional).

NON-GRAPHICAL FEATURES

* Full user control over contour levels.
* Filtering.
* Sophisticated multiple surface operations.
* Trend fitting and analysis (optional).
* Volumetric and areal calculations (optional).

SPECIALISED FEATURES

* Faulting. Contours run right up to the fault lines. This feature may be combined with volumetric calculations.

USER INTERFACE

Two modes of operation are available - totally interactive execution, or interactive set-up for batch execution. Default answers are available to most questions. In addition in ZMAP version 2 interactive editing is available for handling raw data, gridded values and graphical output. The user can reshape contours, faults, etc. 'to correct errors or problems in graphic displays of surfaces' and add new graphical information manually.

COMMENTS

This package is interesting from the point of view of the contouring algorithm used, which appears more advanced than the others found in packages examined in this chapter.

The interactive editing facility is also unusual. It is clear that such a powerful facility could prove very dangerous if not handled with extreme caution. For experts only.

As in the case of the Surface II package, the variety of possible approaches to interpolation could prove counter productive.

## (8) S.D.L. (Surface Display Library)

Dynamic Graphics Inc, Berkeley CA

### TYPE OF PACKAGE

The package produces displays of 2D surfaces in the form of contour plots, perspective block diagrams and projected contour displays. The vendors also supply related packages such as the Surface Gridding Library (SGL) for interpolation and Interactive Surface Modelling (ISM) - an interactive front end for SDL/SGL.

### METHOD OF INTERPOLATION USED

Unusually the interpolation method supplied (in SGL, not SDL) is a global one. Besides being inappropriate in most examples, such a method is likely to be expensive (and possibly ill-conditioned) in large examples.

### CONTOURING ALGORITHM PROVIDED

Not known - but probably piecewise linear (the package 'routinely copes' with grids of over 100,000 cells).

### LABELLING POLICY

The package, 'considers several factors as it labels contour lines'. Labels are only placed where room exists for them, and they follow the orientation of contours.

### OTHER GRAPHICAL FEATURES

* Contour thinning - user-specified interval.
* Hachures.
* 'High' and 'Low' symbols, and/or values at stationary points may be plotted.

NON-GRAPHICAL FEATURES

* Volumetric calculations - but in a separate package.

SPECIALISED FEATURES

* Faulting capability: the package 'routinely copes' with over 2,000 fault segments. Contours are plotted right up to the fault traces. Once again, a separate package is required for this.

USER INTERFACE

Not known, but ISM (interactive front end) available.

COMMENTS

Expensive (see above).

(9) MCS (Mapping-Contouring system)

Scientific Computer Applications, Inc

Tulsa, Oklahoma

TYPE OF PACKAGE

Offers interpolation, contouring and perspective block diagrams.

INTERPOLATION METHOD USED

Similar to the methods employed by GPCP-II and Surface II Graphics.

## CONTOURING ALGORITHM PROVIDED

Two alternative means of contouring from random data are provided: (i) The user may interpolate to a rectangular grid and then use a piecewise linear contouring algorithm. Or (ii), if it is sufficient to plot only within the convex hull of the data sites contouring may be carried out directly:- first the data sites are triangulated so the resulting triangles are 'as nearly equilateral as possible' (this presumably is the Delaunay triangulation: see Green and Sibson (1979)); a subgrid is then formed across each triangle by the construction of equidistant sets of lines parallel with the triangle's sides; values at subgrid vertices are calculated using 'hyperbolic type functions' - on average depending on the values and gradient estimates at six neighbouring points; and finally contours are traced across subgrids by inverse linear interpolation.

## LABELLING POLICY

Labels follow contour orientation, but example plots highlight a tendency of the algorithm to place labels in close pairs separated by relatively large distances.

## OTHER GRAPHICAL FEATURES

* Hachures.

## NON-GRAPHICAL FEATURES

* Volumetric calculations - using the triangular piecewise linear approximation.
* Least squares trend and residual analysis.

SPECIALISED FEATURES

* Faulting.
* 'Multi Surface Contouring':- in examples in which geological formations of fairly constant thickness have folded, information on the depth of one formation is used to help to determine the depth or thickness of another formation, using interpolation and extrapolation.

COMMENTS

Contouring algorithm (i) can produce very angular contours in areas where data is sparse. Contouring algorithm (ii) is similar to the one used by GPCP-II, but subgrids over triangles rather than rectangles.

## 7.4 Packages written for single organisations

ECMWF Contouring Package   The SCD Graphics Utilities

SRC Rutherford Laboratories contouring package

The packages examined in this section represent a small selection of contouring packages written by organisations wholly or primarily for internal use. The possible reasons why an organisation might decide to 'go it alone' in this way rather than purchase a contouring package are numerous:- a package might be written internally for financial reasons; to incorporate specialised facilities not available in a single existing package; to obtain as high as possible a degree of optimisation on the installation's hardware; to integrate the contouring process into an existing graphics framework; or for a combination of these reasons and others. The packages which we describe are used by the meteorological community on both sides of the Atlantic, and by U.K. Government Research Laboratories.

## (10) ECMWF Contouring Package

European Centre for Medium-Range Weather Forecasts

(Petersen, 1980)

### TYPE OF PACKAGE

Contouring from values on a rectangular grid, plus plotting of map projections and various specialised meteorological indicators.

### CONTOURING ALGORITHM USED

A simple piecewise linear method, as described by Dayhoff (1963). Grid cells are divided into four triangles prior to the piecewise linear fit. Contours may optionally be smoothed by the fitting of parametric cubic functions. A more primitive piecewise linear algorithm, which contours cells systematically with no linking, is also provided.

### LABELLING POLICY

Labels follow contour orientation - apparently with no effort to avoid labelling in areas of high curvature or high contour density. The user controls the number of decimal places (or this may be selected automatically) and character thickness.

### OTHER GRAPHICAL FEATURES

* Contour suppression - where surface heights lie outside a user-specified range.
* 'Composite' linestyles - the user may specify one linestyle below/above/at a particular level and another at remaining levels.

NON-GRAPHICAL FEATURES

* Contour levels may be chosen by the user or automatically - but always at regular intervals.

SPECIALISED FEATURES

* The ability to superimpose a variety of coastline projections onto contour maps.
* Suppression of contouring to an octagonal area as a rough approximation to a circle.
* Plotting of 'highs' and 'lows' (local maxima and minima respectively) - either using symbols or by plotting actual heights, or both.
* Plotting of various other meteorological indicators.

USER INTERFACE

Default values are set for most parameters, which may be changed by simple subroutine calls. A relatively simple sequence of subroutine calls is required to produce a complete contour plot.

COMMENTS

All plots at this installation are produced on a raster device (electrostatic plotter) following vector to raster conversion. The package is a fairly undistinguished one, and is being replaced by CONICON and a fast low quality package still to be selected at the time of writing.

## (11) The SCD Graphics Utilities

National Center for Atmospheric Research, Boulder, Colorado

(McArthur (1981))

TYPE OF PACKAGE

A general graphics package which includes interpolation and contouring capabilities.

INTERPOLATION METHOD PROVIDED

The $C^1$ surface algorithm of Lawson (1977), as refined by Akima (1978).

CONTOURING ALGORITHM USED

The simplest piecewise linear method, with no internal subdivision of grid cells. Contour smoothing is available using the method of splines under tension. The user has control over the tension factor which determines the smoothness of the curves.

LABELLING POLICY

Labels follow contour orientation, and their size and format are under user control. The most sophisticated routine protects a rectangular area round each label from being touched by all contours.

OTHER GRAPHICAL FEATURES

* Contour suppression - no contouring when grid values lie outside a specified range.

NON-GRAPHICAL FEATURES

* The user has full control over the contour levels which are plotted.

SPECIALISED FEATURES

* Plotting of 'highs' and 'lows', both by symbols and values. The user controls the size of the labels.
* Plotting of other meteorological indicators and coastline projections.

USER INTERFACE

A complete contour plot can be produced by a single subroutine call. Other subroutine calls may be executed prior to contouring to change default parameter values.

COMMENTS

A fairly uninteresting package with a poor contouring algorithm.

(12) <u>SRC Rutherford Laboratories Contouring Package</u>    <u>Atlas Computing Division</u> (Sutcliffe (1976))

TYPE OF PACKAGE

Contouring only, from values on a rectangular grid. The grid may be irregular and/or skewed.

CONTOURING ALGORITHMS USED

A choice of two piecewise linear algorithms is provided, both due to Heap (1974). The first carries out no internal cell subdivision, while the second divides grid cells into four triangles in the usual way. No means of contour smoothing is provided.

## LABELLING POLICY

Labelling is by upright integer value only, with a key provided for interpretation. Gaps are not left in contours where labels occur. The user controls the frequency of labels on contours (in terms of the number of steps between labels), but each section of contour must have at least one label. The user may alter the character sizes used for labelling and in the key table independently.

## OTHER GRAPHICAL FEATURES

* Contouring may be suppressed outside a polygon whose sides must comprise diagonals or edges of grid squares. This feature is available for square grids only.

## NON GRAPHICAL FEATURES

* The user has full control over the contour levels which are plotted - or they may be set automatically.

## SPECIALISED FEATURES

None.

## USER INTERFACE

Default values of parameters are set to minimise the length of argument lists and a single subroutine call is sufficient to create a complete contour map. Extra subroutine calls are required to change parameter values.

## COMMENTS

A very primitive package, similar to the contouring facility in the NAG Graphical Supplement, but even less sophisticated.

## 7.5 Packages which specialise in a subset of the contouring process

DISSPLA Mapez

In this final section we consider a pair of packages which have little in common besides the fact that each specialises in a subset of the contouring process - the DISSPLA package is primarily intended for displaying in an attractive way the contours produced by an arbitrary contouring algorithm, while ZMAP provides an interactive user interface for use on top of a relatively sophisticated contouring package. Assessment of the ZMAP package has been performed on the basis of sales literature alone.

### (13) DISSPLA

ISSCO Corp, San Diego, CA

TYPE OF PACKAGE

A general graphics package offering a wide range of facilities including contouring.

CONTOURING ALGORITHM USED

The package provides a simple piecewise linear algorithm for contouring from a rectangular grid of values, but is primarily intended for the presentation of contours produced by other packages.

LABELLING POLICY

By storing all the 'contours' simultaneously in labelled Common, the package ensures that pairs of labels can never overlap and that contours respect the positions of all labels. Labels follow contour orientation and the user controls minimum inter-label distances and the degree of curvature which is tolerated where labels occur.

OTHER GRAPHICAL FEATURES

* Thinning - the user specifies the minimum distance allowed between adjacent contours, and priority levels for contours which enable the package to determine which contours should be omitted where thinning occurs.

USER INTERFACE

In the form of Fortran subroutine calls, with no default values set. Can be relatively complicated if a wide variety of linestyles is used, as a single call is required to establish each style in the cycle.

COMMENTS

Results are of a high quality, but only at the cost of severe penalties in terms of both CPU and memory usage. Such penalties are however probably unavoidable if attractive results are to be derived from input data of this nature, since knowledge of the surface being contoured, which could have aided both thinning and label positioning, has been discarded.

A further but relatively trivial defect in the package is that labelling and linestyle definition are not controlled independently.

(14) Mapez: Interactive contour plotting Interface

Zycor Inc, Austin, Texas

TYPE OF PACKAGE

An interactive front end for the CPS-1 package (see section 7.3).

## USER INTERFACE

The package provides, it is claimed, a 'user friendly' interactive interface which simplifies use of the CPS-1 package. As we saw in section 7.4, the latter package offers a wide range of facilities which the inexperienced user might require a considerable amount of time to master. It is therefore conceivable that in a commercial environment the savings in training time derived from the acquisition of a package such as MAPEZ might justify its purchase.

The manufacturers claim that the package eliminates errors and reduces the time needed to set up a new CPS-1 run. It is structured to be suitable for use by experienced and inexperienced users alike:- a limited explanation is provided at points requiring a response, which may be supplemented by using a 'HELP' command. It is also possible to program the package so that the simple user is exposed only to a subset of the logical decision points.

## COMMENTS

Marketed by the distributors of the ZMAP package (see Section 7.4); not by the distributors of CPS-1!

# CHAPTER 8

# CONCLUSIONS AND SUGGESTIONS FOR FURTHER WORK

The review of existing contouring methods which opened this thesis was presented primarily with the intention of highlighting the defects of these methods, and of persuading the reader of the need for a new contouring technique capable of combining convincing and accurate representation of contours with low execution costs. The subsequent investigations and development of the seamed quadratic method have convinced the author that such a method has indeed been discovered; moreover mathematical understanding of the method now stands at a level which is unrivalled by most of its competitors.

A clear indication of the quality of the method was given as early as in Chapter 2 (Figure 2.4), where a single seamed quadratic element was demonstrated to be capable of representing a relatively complex surface with contours of perfect visual smoothness. However the broad range of examples presented in Chapter 4, which include comparisons with other contouring methods, provides a much more powerful argument which, it is hoped, should prove sufficient to convince the most sceptical of readers.

At no point, however, have we claimed that the method is in any way optimal (except perhaps within the category of methods based on rectangular seamed quadratic finite elements), and a few problems involved in using the seamed quadratic element have emerged which should be recognised: firstly, like all contouring methods, it will produce anomalous results when the Implicit Function Theorem breaks down or comes close to breakdown; it has also (see Section 4.3) been shown to produce spurious peaks in areas where the slope of a surface decreases exceedingly rapidly; and finally, when combined with any interpolation method, it will not in general produce a surface which respects exactly the original data values. However the first two of these defects will not affect the vast majority of examples, and we have suggested (Section 6.5) a means of minimising the effects of the third (though we have not had time to put this into practice).

In Chapter 3 we explained how the seamed quadratic contouring method was implemented as the CONICON package and developed to a high degree (the amount of work involved in this development being considerably under-represented by the length of the chapter). It is believed that, as a result of these efforts, the features of the package (crosshatching especially) compare favourably with those of most packages examined in Chapter 7, but that there is still much potential for improvement to be made. If the piecewise quadratic contouring method introduced in this thesis is not to be overlooked for a number of years to come, it is particularly important that the reliability, ease of use, efficiency and range of features of the CONICON package are all developed to as high a level as possible.

Clearly it is highly desirable that those improvements made to the ECMWF version of the package should become standard. After this, perhaps the most pressing need is that the package should become more 'user friendly', but several other areas with much potential for improvement exist: the contour suppression feature might be extended to incorporate contouring within an arbitrary polygon, and thinning, hachures and cross sectional views might also usefully be included.

In the area of efficiency a number of aspects of the package could be improved considerably. For example, crosshatching algorithm A is a prime contender for savings: we know that, in the largest examples, where limitations on the availability of memory can become a problem, a contour across a triangle is typically represented by a single straight line segment; the amount of memory used by this algorithm could therefore be cut dramatically by storing contour 'vertices' themselves rather than the twelve reals per triangle which currently need to be stored. As a by-product of this the calculation of intersections between contours and hatching lines would be simplified, improving both CPU usage and numerical stability. A further possibility for memory savings relates to the arrays XY and CONT (see Chapter 3 for details): it might be possible (probably at the cost of slightly increased CPU usage) to dispense with the latter array, which merely duplicates (though in a more convenient order) information held in the former. A number of more minor possibilities for program optimisation also exist. Such reductions in memory usage would probably be a prerequisite for large-scale use of the package on mini-computers, where an increasingly large amount of graphical work is carried out.

The error analysis reported in Chapter 5 has provided us with a relatively high degree of understanding of the seamed quadratic method, which compares favourably with the general lack of mathematical knowledge of rival contouring methods. The results obtained should prove useful both in the design and analysis of piecewise quadratic contour maps. For the purposes of comparison it would be beneficial to carry out similar error analyses on other seamed quadratic elements, and to examine piecewise linear approximation and the piecewise cubic element introduced in Chapter 2 in a similar manner.

The investigations conducted in Chapter 6 of locally adaptive contouring methods based on the use of the seamed quadratic element have probably raised more questions than they have answered. The need to improve the computational implementation of grid construction is probably of little urgency, for although we have presented only a first attempt at a solution this involves use of an insignificant amount of resources compared with those needed by the contouring part of the process. However much work is still needed to investigate the many possible splitting criteria, a number of which were suggested in the discussion which forms the final part of the chapter. We must conclude that at present local subdivision using the splitting rules devised and tested by the author is not practicable, given the difficulties of choosing parameter values 'blindly'; however if some automatic or semi-automatic means of selection could be devised these techniques might prove very useful - in one particular example in our investigations (see Figures 6.14 and 6.15) the splitting rule appears ideal. However a number of possible splitting criteria remain completely untried, and investigation of their capabilities should be an interesting - and possibly rewarding - venture.

The third derivative estimate (based on vertex values and gradients) derived in Chapter 6, although arguably defective in the context of adaptive local cell subdivision, might well be of more use in the design of regular grids, though we have not investigated this: the poor accuracy of the estimate is unlikely to be a serious problem in this context, and of course its other known defect - that it apparently does not guarantee termination of the splitting process - is not applicable here.

Our final chapter, which compares 14 existing contouring packages, is unavoidably such a superficial investigation that it is really only possible to draw tentative conclusions about the relative merits of each package. A detailed investigation of the packages, comparing the performance of each one on a number of standard data sets, would probably involve a complete Ph.D. project in itself. Whether it would be worthy of such a project is arguable:- certainly the results of such a comparison would be of considerable interest to potential users, but the apparent similarity of contouring algorithms incorporated in these packages indicates that an investigation of this nature would probably be of little interest from a scientific point of view.

We end then with the conclusion that the seamed quadratic method represents a significant advance in the field of automatic contouring which has great potential value. However, if this potential is to be realised and the method widely used then it is extremely important that the CONICON package should present the method in the best possible light. Improvement of CONICON is therefore one of two major areas for further work which we have identified; the other is further investigation of the highly promising area of locally adaptive contouring.

## REFERENCES

AKIMA, H (1978). A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points. *ACM Trans. on Math. Software*, 4, pp.148-159.

APOSTOL, T.M. (1957). *A modern approach to advanced calculus.* Addison-Wesley, Reading, Mass. - Palo Alto - London.

BATCHA, J.P. and REESE, J.R. (1964). Surface determination and automatic contouring for mineral exploration, extraction and processing. *Colorado School of Mines Quarterly,* 59, pp.1-14.

BRENT, R.P. (1974). Algorithm 488. *Communications of the ACM*, 17, pp.704-706.

BUTLAND, J. (1980). A method of interpolating reasonably-shaped curves through any data. *Proceedings of Computer Graphics 80*, pp.409-422. Online publications.

CALCOMP APPLICATIONS SOFTWARE (1972). *GPCP-II: A General Purpose Contouring Progam. User's Manual.* Calcomp, Anaheim, California.

CALCOMP APPLICATIONS SOFTWARE (1974). *Supplement to GPCP-II: A General Purpose Contouring Program. User's Manual.* Calcomp, Anaheim, California.

CLOUGH, R.W. and TOCHER, J.L. (1965). Finite element stiffness matrices for analysis of plates in bending. *Proc. Conference on Matrix Methods in Structural Mechanics, Wright-Patterson A.F.B., Ohio, 1965.*

COMPUTER AIDED DESIGN CENTRE. *Ginosurf User Manual. Issue 1.* CADC, Cambridge.

COTTAFAVA, G. and LE MOLI, G. (1969). Automatic Contour Map. *Communications of the ACM,* 12, pp.386-391.

CRANE, C.M. (1972). Contour plotting for functions specified at nodal points of an irregular mesh based on an arbitrary two parameter co-ordinate system. *The Computer Journal,* 15, pp.382-384.

DAYHOFF, M.O. (1963). A Contour-Map program for X-Ray Crystallography. *Communications of the ACM*, 6, pp.620-622.

DELFINER, P. and DELHOMME, J.P. (1975). Optimum interpolation by Kriging. In *'Display and Analysis of Spatial Data'*, pp.96-114, Wiley.

FALCONER, K.J. (1971). A general purpose algorithm for contouring over scattered data points. *NPL Report NAC 6*.

FITCH, J.P. (1982). *CAMAL User's Guide (2nd Edition)*, University of Cambridge Computer Laboratory.

GREEN, P.J. and SIBSON, R. (1978). Computing Dirichlet tessellations in the plane. *The Computer Journal*, 21, pp.168-173.

HEAP, B.R. (1974). Two Fortran contouring routines. *NPL Report NAC 47*.

HEAP, B.R. and PINK, M.G. (1969). Three contouring algorithms. *NPL Report DNAM 81*.

HUNTER, G.M. and STEIGLITZ, K. (1979). Operations on Images using Quad Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-1 (No.2)*, pp.145-153.

ISSCO GRAPHICS (1982). *DISSPLA User's Manual*. ISSCO Corp, San Diego, California.

KLINGER, A. and DYER, C.R. (1976). Experiments on picture representation using regular decomposition. *Computer Graphics and Image Processing*, 5, pp.68-105.

LAWSON, C.L. (1977). Software for $C^1$ Surface Interpolation. In *Mathematical Software III*, pp.161-194. Academic Press.

LODWICK, G.D. and WHITTLE, J. (1970). A technique for automatic contouring field survey data. *Australian Computer Journal*, 2, pp.104-109.

MCARTHUR, G.R. (1981). The SCD Graphics Utilities. *NCAR Technical Note NCAR-TN/166+IA*. NCAR, Boulder, Colorado.

MCCONALOGUE, D.J. (1970). A quasi-intrinsic scheme for passing a smooth curve through a discrete set of points. *The Computer Journal*, 13, pp.392-396.

MCCONALOGUE, D.J. (1971). An automatic French-curve procedure for use with an incremental plotter. *The Computer Journal,* 14, pp.207-209.

MARLOW, S. and POWELL, M.J.D. (1976). A Fortran subroutine for plotting the part of a conic that is inside a given triangle. *UKAEA Harwell Paper AERE-8336,* HMSO London.

MILNE, W.P. (1924). *Homogeneous coordinates.* Edward Arnold and Co. (London).

NUMERICAL ALGORITHMS GROUP (1981). *NAG Graphical Supplement Mark 1.*

NUMERICAL ALGORITHMS GROUP (1982). Routine G05DDA/F. *NAG Fortran Library Manual Mark 9, Volume 6.*

PERCELL, P. (1976). On cubic and quartic Clough-Tocher finite elements. *SIAM Journal of Numerical Analysis,* 13, pp.100-103.

PETERSON, A. (1980). Contouring Package User's Guide (Revision 2). *ECMWF Computer Bulletin B5.2/3(2).*

PHILLIPS, E.G. (1962). *A course of analysis* (2nd. edition), Cambridge University Press.

POWELL, M.J.D. (1974). Piecewise quadratic surface fitting for contour plotting. In *'Software for Numerical Mathematics'* (D.J. Evans ed.), Ch.14, pp.253-271, Academic Press.

POWELL, M.J.D. and SABIN, M.A. (1977). Piecewise quadratic approximation on triangles. *ACM Transactions on Mathematical Software,* 3, pp.316-325.

RITCHIE, S. (1978). *Representation of surfaces by finite elements.* M.Sc. Thesis, University of Calgary.

ROBINSON, E.L. and SCARTON, H.A. (1972) "CONTOR - FORTRAN subroutine to plot smooth contours of a single valued 3-D surface". *J. Comput. Ph.*, 10, p.242.

ROTHWELL, M.A. (1971). A computer program for the construction of pole figures. *J. Appl. Cryst.*, 4, p.494.

SABIN, M.A. (1980). Contouring - A review of methods for scattered data. In *'Mathematical Methods in Computer Graphics and Design'* (ed. K.W. Brodlie) Ch. 3, pp.63-85. Academic Press.

SAMPSON, R.J. (1975, revised 1978). *Surface II Graphics System.* Kansas Geological Survey.

SCHAGEN, I.P. (1979). Interpolation in two dimensions - a new technique. *J. Inst. Maths Applics* 23, pp.53-59.

SCHAGEN, I.P. (1982). Automatic Contouring from Scattered Data Points. *The Computer Journal,* 25, pp.7-11.

SHELL, D.L. (1959). A high-speed sorting procedure. *Communications of the ACM,* 2, pp.30-31.

SHVIDLER, M.I. (1964). *Filtration flows in Heterogeneous Media.* Consultants Bureau, New York (translated from Russian).

SIBSON, R. (1980). *Tile 4 User's Guide.* University of Bath.

SIBSON, R. (1982). A brief description of natural neighbour interpolation. In *'Interpreting Multivariate Data',* (ed. V. Barnett) Ch.2, pp.21-36. Wiley, London.

SIBSON, R. and THOMSON, G.D. (1981). A seamed quadratic element for contouring. *The Computer Journal,* 24, pp.378-382.

SILVERMAN, B.W. (1982). Density Estimation for Univariate and Bivariate Data. In *'Interpreting Multivariate data'* (ed. V. Barnett). Ch.3, pp.37-53. Wiley, London.

SUTCLIFFE, D.C. (1976). Contouring. *Graphics User Note 1.* SRC Rutherford Laboratory, Atlas Computing Division.

SUTCLIFFE, D.C. (1980). Contouring over rectangular and skewed rectangular grids - an introduction. In *'Mathematical Methods in Computer Graphics and design'* (ed. K.W. Brodlie). Ch.2, pp.39-62. Academic Press.

WAHBA, G. (1979). How to smooth curves and surfaces with splines and cross-validation. *University of Wisconsin Department of Statistics Technical Report no.555.*

WAHBA, G. and WOLD, S. (1975). A completely automatic French curve: fitting spline functions by cross-validation. *Communications in Statistics,* 4, pp.1-7.

WARNOCK, J.E. (1969). A hidden-surface algorithm for computer generated pictures. *University of Utah Computer Science Department Report TR4-15.*

WOODWARK, J.R. (1982). The explicit quad tree as a structure for computer graphics. *The Computer Journal,* 25, pp.235-238.

ZACHRISEN, M. (1979). 'Surrender' - A subroutine package for rendering bivariate surfaces. *Runit (Computing Centre at the University of Trondheim) report STF14 A79020.*

APPENDIX A

CONICON DOCUMENTATION

```
 ******
**    **
**                             **
**         ******   * *****          ******   ******   * *****
**        **    **  **    **  ***   **       **    **  **    **
**        **    **  **    **   **   **       **    **  **    **
**    **  **    **  **    **   **   **       **    **  **    **
 ******    ******   **    **  ****   ******   ******   **    **

                             ****
                            *   **
                                 **
                                **
                               **
                              **
                             **
                            *******
```

# CONICON 2 USER'S GUIDE

# FORTRAN CONICON 2 CREATED 1 AUGUST 1982

A PACKAGE FOR THE PRODUCTION OF HIGH QUALITY CONTOUR PLOTS

Originators:

Professor R. Sibson & Mr C.D. Thomson
School of Mathematics
University of Bath
Claverton Down
BATH, Avon
ENGLAND BA2 7AY
Telephone Bath (0225) 61244
Telex 449097

This library was prepared as part of the work of an SSRC-supported project "Development of Statistical Methods for Analysing Spatial Data" directed at the University of Bath by Professor R. Sibson in the period 1977-82.

## INTRODUCTION

The purpose of the CONICON package is to provide a range of sophisticated high-quality contour plotting facilities for use in conjunction with a vector graphics device. The emphasis is on the quality and accuracy of the maps produced by the package, but the techniques used to achieve this prove to be efficient in terms of the computational load they impose.

The package is written in a subset of ISO FORTRAN contained within FORTRAN77, and no difficulty should be experienced in running it under any normal FORTRAN system. However, memory requirements may limit its useability on very small computers. All arithmetic carried out within the package is single precision, and reliable results should be obtainable in 32-bit arithmetic or better.

The package consists of FORTRAN subprograms; the user normally chooses one of a small number of master routines to produce the desired contour map and may also wish to call various utility routines to assist in setting up data for the master routine. Each such routine has to be provided by the user with height and gradient information at a grid of points on the surface to be contoured, together with control information to define the map that is required, and uninitialised workspace. The map is constructed by calls to graphics primitives (eg "draw a line to (x,y)") which are themselves expressed as FORTRAN subprograms; it is the responsibility of the installer of the package to provide implementations of these primitives, which are defined in detail below, to link CONICON to a graphics driver.

## MATHEMATICAL BACKGROUND

It is possible to use the CONICON package without reading this section, but users who intend to produce contour plots at all regularly are strongly recommended to do so in order to understand properly the nontrivial nature of the contouring process.

Suppose (x,y) are cartesian coordinates giving position in the plane (eastings and northings in conventional cartography), and f is some real function of position; for example, f(x,y) might be topographical height, ground level atmospheric pressure, optical density on a photographic plate, and so on, measured at the point (x,y). Fix a value h, and consider all those points in the plane such that $f(x,y) = h$. This is the contour at level h. It may be an empty set (if h is not a value taken by f), and even if nonempty, it may in general be in a mathematical sense a very unpleasant set. Usually, however, it consists of a number of smoothly curved lines, the contour lines or isolines at level h. It is these lines that contouring packages attempt to draw on a graphics device. One of the reasons why contouring is a difficult task computationally is that this familiar "nice" case depends on a number of mathematical conditions holding. Shorn of various technical caveats, the main condition is that a well-defined contour line through the point (x,y) exists if f has nonzero gradient at (x,y), and the contour line is then differentiable to the extent that f is at and near (x,y). Those wishing to understand the full details should read a textbook on the analysis of functions of several real variables, and refer in particular to the section dealing with the Implicit Function Theorem. In practice, the contours of a function that is not continuous will themselves display discontinuities, as, for example, at a cliff-edge; and if the function is not differentiable, the contours may have "corners", as, for example, at the bottom of a V-shaped valley. If the function is continuously differentiable (is of class C1), then so will the contours be. A continuously differentiable curve is visually smooth: the eye is very good at detecting discontinuities of value and of gradient, but very bad at detecting discontinuities of curvature and higher derivatives. Most contouring packages, CONICON included, are designed to draw the contours of continuously differentiable functions. Even in this case, good behaviour of the contour lines depends on non-zero-ness of the gradient of f. The most familiar example of the breakdown of this condition is at a saddlepoint, where the contour at exactly the level of the saddlepoint looks locally like two straight lines crossing. Behaviour worse than this can happen only at a point where higher derivatives vanish as well as the gradient, and it is only under these circumstances that a contour line can touch, rather than cross, itself. Contour lines associated with different levels can never, for obvious reasons, touch or cross one another.

Even when the function f has a known and explicit mathematical form, it is seldom possible to solve explicitly the equation $f(x,y) = h$ which determines the contours. Thus contouring has to be an indirect process in practice. Some contouring techniques attempt to approximate the contours of the function f as it stands. This approach leads to serious difficulties over generating each contour once and once only, and moreover the approximation procedure requires "random access" to values of f at all points in the plane, which in many cases makes it so costly to carry out computationally that a low standard of accuracy has to be accepted, often leading to contours which even for smooth functions display unacceptable, unlikely, or impossible features such as visible sharp bends, cusps, self-contacts, or even crossing of contours at different levels.

The alternative approach is to approximate f by a function g whose mathematical form is such as to allow the explicit solution of the equation g(x,y) = h, and then to draw the contours of g. Because these have an explicit form, they can be drawn to the limits of resolution of the graphics device, or of the eye if that is coarser; their accuracy depends on how good an approximation g is to f. Since that approximation is made once-for-all, the evaluations of f are predetermined in number and position - usually they lie on a square grid whose fineness is the control over approximation accuracy. This latter approach is the one used in CONICON. It has one inherent disadvantage, which is as follows. Quite often the function f is itself constructed by interpolation by some method between values observed at an irregular scatter of positions. Note, incidentally, that any such interpolation procedure is quite distinct from the process of constructing contours, and methods which attempt to conflate the two procedures should be viewed with suspicion. When f is such an interpolant, it will generally be the case that g cannot coincide with f at all the points where observations have been taken, and if a contour level falls between the value of g and the observed value at such a point, then the contour will pass on the wrong side of the point. In practice this seldom happens, and when it does so the vertical error is small and hence the horizontal error usually is too. If it is found to be unacceptable, then for a given set of contour levels it can be avoided by improving the accuracy with which g approximates f, at a cost in computational load.

The difficulty in implementing the approximating-function approach to contouring has historically been that of choosing a suitable class of functions G from which g is to be selected. The requirements are that G must contain enough functions to allow good approximation to f; that the functions in G should allow explicit solution of the contour equation g(x,y) = h; and that the functions in G should be smooth enough to have smooth contour lines, which means in practice that they must be continuously differentiable. The technique used in many packages is to evaluate f on a (usually square) grid of points, and then construct g across each cell of the grid, for example as a bilinear function matching the values of f at the four corners of the cell. Although such functions are continuously differentiable within each cell, they are only continuous across cell boundaries and do not have continuous derivative there, so their contours consist of sections of smooth curve with sharp bends, leading to maps of unacceptable quality. Most of the other approximant functions which have been proposed display similar defects.

The main novelty of the CONICON package is its use of a new class of approximant functions G, constructed from evaluations not only of f but also of its gradient on a grid (square in CONICON 2, although this is not an inherent limitation). Experience shows that in many cases it is possible to evaluate the gradient of a function at very little additional cost to that involved in evaluating the function itself. Where this is not possible, the gradient at each grid point can easily be estimated from the values there and at neighbouring points, and the only cost is one of accuracy since the use of the true gradient allows a higher-order fit between g and f. A utility to estimate gradients from value-only data is provided in CONICON 2. A value and gradient is accordingly assumed to be available at each vertex of each grid cell. The value and gradient information at the four corners of a cell is sufficient to permit the construction of a function across that cell which joins on to the functions in neighbouring cells with continuity of gradient as well as value. Such a function is called a C1-conforming finite element. Clearly a function constructed in this way will have smooth contour lines provided that the function is continuously differentiable within each cell. Because of the value-and-gradient matching to f, the quality of the approximation is high, and accurate contour maps can be produced without the need to use a very fine grid.

The finite element that is used has to permit the explicit solution of the contour equation g(x,y) = h, and yet has to offer enough flexibility to match value and gradient at the corners of a square. Quadratic functions are functions for which g(x,y) = h can be solved; the resultant contours are conic sections. First-degree ("linear") functions do not offer enough flexibility - their contours are straight lines - and higher degree functions do not in practice permit the solution of g(x,y) = h in any helpful parametric form; this also seems to be true of non-polynomial functions. However, quadratics in two variables are a six-parameter family and cannot be expected to match twelve data values (value and two components of gradient at the four corners of the square cell). It is necessary to break the cell up into triangular patches with a separate quadratic on each and with joins across internal seam lines retaining continuity of the function and its gradient. The resulting construct is called a seamed quadratic finite element. In addition to matching the data at the corners, it is necessary for the element to satisfy conditions along its edges which ensure a smooth join to neighbouring elements. The authors have proposed a novel element for this purpose, and that is what CONICON uses. The element is described in the following paper:

Sibson, R., and Thomson, G.D. "A seamed quadratic element for contouring"
The Computer Journal 24 (1981) pp. 378-382.

A detailed account of its properties, including error bounds, is given in the following thesis:

Thomson, G.D. "Automatic contouring by piecewise quadratic approximation"
University of Bath PhD Thesis (1982).

The Sibson-Thomson element divides the square (or rectangle) into sixteen triangles. First the cell is halved horizontally and vertically, then each quarter-cell is divided into four by its diagonals. Although this sounds complicated, it is in fact a parsimonious solution, in that no degrees of freedom remain after the values and gradients at the grid points have been matched and the conformability conditions along the edges have been satisfied. The construction of the contours on such a system is a complicated and numerically quite delicate task, but it is much more efficient than might at first sight be supposed, mainly because good approximate bounds are available which greatly facilitate the location of those cells, and subsequently of those triangles, where a contour may lie. The contours themselves consist of pieces of conic section (ellipse, parabola, hyperbola) joined continuously differentiably together. These conic segments, being described parametrically, can be drawn to any desired degree of accuracy on an actual graphics device.

## FEATURES OF THE PACKAGE

### (1) Choice of contour heights

The user, if he wishes, may specify all contour heights himself and these need not be at regular intervals. However he has the alternative of allowing the package to choose contour levels for him. This may be done completely automatically, or the user may specify a pair of heights between which all contours should lie. The user must always specify the number of contours himself. Note also that in order to keep contour heights at reasonably round numbers, automatic choice of contour levels will not always result in the precise number of contours requested being plotted.

### (2) Annotation of contours

If he wishes, the user may request the package to label the contours in his plot, and this will be done with suitably-sized gaps being left in the contours where labels occur. If this feature is used, it is not necessary to label all contours - the user has the option of labelling every nth contour and leaving the remainder unlabelled.

### (3) Thick-line contours

Thick-line contours are an additional feature. The user exercises the same degree of control over their positions as he does with annotated contours.

### (4) Crosshatching

A distinctive feature of the CONICON package is the ability to crosshatch (i.e. "shade in") the area between pairs of contours (also above or below specified contours). Styles of crosshatching vary from the most basic straight-line crosshatching to sophisticated styles such as digits from 0 to 9, honeycomb, basketwork and tree styles for cartographic use.

Incorporated in the package are two separate and fundamentally different crosshatching algorithms, neither of which can be considered superior to the other in all circumstances. The user's preferred choice of algorithm will depend upon such factors as the number of grid cells in the plot and the particular styles of crosshatching to be used. A discussion on the relative merits of these algorithms is given below in the section entitled "Selection of crosshatching algorithm."

### (5) Local suppression of contour plotting.

CONICON always requires the user to specify an M * N grid of surface heights and gradients; however the user may instruct the package to suppress plotting within a set of grid squares of his choice. It is therefore possible to leave "holes" in the plot and to construct plots with non-rectangular boundaries.

In many cases the user may wish to restrict contour plotting to the part of the grid lying within a polygonal boundary, and in a number of these examples the polygon will be a convex one. A subroutine is therefore provided which selects for the user those cells of the grid which lie within such a region as specified by the user. A further subroutine may be used to plot the boundary of the area which will then be contoured.

(6) Combinations of features (1) - (5)

The user may combine any of the features described in (1) - (5) simultaneously, with the exception of features (4) and (5) which cannot be used together in the present implementation. In particular, if the options of annotation and crosshatching are chosen together, the crosshatching lines will leave enough space for the labels to be seen clearly.

(7) Plotting the gradient of a function

Besides enabling the user to produce standard contour plots of a function, CONICON also incorporates a facility for the creation of contour plots of the squared magnitude of the gradient (that is, the sum of squares of the partial derivatives in x and y directions) of that function. As with standard CONICON plots, the contours plotted using this facility are not the true (gradient) contours of the surface, but are the contours (of the gradient) of the piecewise quadratic approximant function (see Sibson and Thomson (1981) for details). This feature is included in the package as an aid to the understanding of standard CONICON plots:- if it indicates the existence of large areas throughout which the gradient of the function is very small, then the user should regard contours of the function itself in such areas with some scepticism. In these areas the Implicit Function Theorem, which is the foundation of contouring, comes close to breaking down and contours may start to display anomalous behaviour.

Note that gradient contours, unlike other contours produced by CONICON, need not be smooth throughout and may have visible "corners" in some areas. This is because the squared magnitude of the gradient of the approximant is continuous but need not have continuous derivatives across seam lines; the "corners" are not an error.

Features (1), (2), (3) and (5) may all be used in combination with this facility.

(8) Plotting of stationary points

A further useful feature offered by CONICON is its capacity for the marking of stationary points (local maxima, local minima and saddle points) of a function. The points plotted are not of course true stationary points of the function being contoured, but are stationary points of the piecewise quadratic approximant function and will usually be a good approximation to the true stationary points.

The locations of stationary points of each type may be indicated by separate symbols selected by the user; for example, in meteorological applications the user may decide to indicate the presence of local maxima and minima by "H"s and "L"s respectively, and to suppress labelling of saddle points.

It is anticipated that this feature will normally be used in conjunction with standard contour plots, but it may also be used in combination with gradient plots or indeed independently of any contour plots. The occurrence of several stationary points in a cluster is fairly common, and reflects a suggestion in the gridded values of the function and its derivatives that some higher derivatives may vanish at or near the cluster. Users should not automatically assume that an error has occurred when several stationary points appear together.

## SELECTION OF CROSSHATCHING ALGORITHM

As has been mentioned above, the CONICON package allows the user a choice of two different crosshatching algorithms, which we shall refer to as algorithm A and algorithm B. If the user wishes to use algorithm A he should call subroutine CONXA1 or CONXA2; to use algorithm B he should call subroutine CONXB1 or CONXB2. In order to decide which is the more suitable for his purposes, the user should be aware of the relative merits of algorithms A and B, so we give here a brief discussion of this matter.

If algorithm A is used then the user must choose a single style of hatching to be plotted between each adjacent pair of contours, plus styles for above and below the highest and lowest contours respectively (The option of leaving any of these areas blank is of course available). The algorithm is only capable of hatching these areas one at a time, and therefore cannot take advantage of situations such as two neighbouring styles of crosshatching being identical (in which case it would be more efficient to treat the two areas as a single area and crosshatch them simultaneously).

Algorithm B allows more flexibility. The number of styles of crosshatching chosen is completely independent of the number of contours plotted (indeed it is not necessary to plot any contours); and each band of crosshatching is chosen independently of contour heights and of the other bands of crosshatching. Thus superposition of crosshatchings becomes possible. This is particularly useful if the user wishes the crosshatching to darken progressively as higher levels of the surface are reached, and also enables the user to create styles of crosshatching not available when algorithm A is used.

Unfortunately algorithm B is in general less efficient than algorithm A, in the sense that if both are given identical tasks (i.e. tasks which A is capable of carrying out) then algorithm A will usually take quite considerably less time in completing the task than algorithm B. However, in the case mentioned above where the user wishes the intensity of crosshatching to increase progressively as the height of the surface increases, algorithm B is usually considerably faster than algorithm A if the opportunity of superimposing different bands of crosshatching is taken up. AUTOXH, a subroutine which automatically sets values of hatching parameters for algorithm B in such cases, is documented below.

Another important factor which must be taken into consideration is storage requirements. The use of algorithm B involves little additional storage space on top of that used to produce the map's contours. On the other hand algorithm A requires a number of extra arrays whose length depends on the number of elements in the grid as well as the complexity of the contours being produced (see notes on CONXA1 for details). It is possible that for some plots on some systems the total storage space required may be prohibitively large. In such cases the only alternative is to use algorithm B.

Finally, if the user wishes to combine crosshatching with annotation, he should note that algorithm B hatches the plot after construction of all contours and is therefore able to leave a small rectangular area around every label unmarked, whereas algorithm A calculates and plots each band of crosshatching before the construction of all but one of the contours above the level of that band, and cannot predict label positions on contours which have not yet been plotted.

It is hoped that these notes will help the user to assess which of algorithms A and B is better suited to his individual needs. However, if a large number of plots is to be produced then he is recommended to experiment with both before deciding which is to be preferred.

Users may wish to know that the efficiency of algorithm A and the flexibility of algorithm B could in principle be combined, but this would involve very heavy storage penalties indeed, to the extent that it has not been considered worthwhile to offer this option in the package.

## GRAPHICS INTERFACE

Included in the CONICON package are calls to a number of simple graphics routines, which the user will have to supply to interface with the graphics system on his computer. These routines must conform to the following specifications:

### 1) SUBROUTINE PLTON(XMIN,XMAX,YMIN,YMAX)

This routine should initialise a frame of the plot and set up a rectangular plot window, the southwest corner of which is the point (XMIN,YMIN) and the northeast corner of which is the point (XMAX,YMAX). The scales should be equal in the x and y directions. All the plotting done by the routines described below will fall within this rectangle.

### 2) SUBROUTINE PLTMOV(X,Y)

This subroutine should move the plot position invisibly to the point (X,Y). The first change of plot position after a call made by the graphics routines to PLTON will be by a call to PLTMOV.

### 3) SUBROUTINE PLTLIN(X,Y)

This subroutine should draw a straight line in the current line style and colour from the last plot position to the point (X,Y), where the plot position should be left.

### 4) SUBROUTINE PLTFAT(X,Y,H)

This subroutine should draw a (generally thin) rectangle from the current plot position to the point (X,Y) in the current logical pen colour and leave the current plot position at the point (X,Y). The rectangle should be 2.0*H units thick. The purpose of this routine is to give the impression of a thick line and it is therefore acceptable (and perhaps desirable) to plot a straight line from the current plotter position to (X,Y) before plotting the rectangle. In this way the pen will be in the correct position immediately after the rectangle has been plotted.

### 5) SUBROUTINE PLTFIG(FNUM,X,Y,NFRAC)

This subroutine should print the floating point number stored in FNUM from left to right so that its centre is at the point (X,Y). The integer NFRAC should set the number of digits to appear after the decimal point of the number. If NFRAC is zero the decimal point should be suppressed. The current plot position on return from this subroutine should be the point (X,Y)

## 6) SUBROUTINE PLTCCS(CHW,CHH)

This subroutine should return, in user coordinates, the width and height respectively of the rectangular area in which a single character is plotted by subroutine PLTFIG (including spaces to the next character on the right and to the next line of characters above). The current plotter position should be unchanged on exit.

## 7) SUBROUTINE PLTPEN(IPEN)

This subroutine should set the logical pen colour to the type indicated by IPEN and should leave the current plotter position unchanged. A value of IPEN = 0 should correspond to solid lines. A negative value of IPEN should suppress plotting.

## 8) SUBROUTINE PLTSYM(ISYM)

This routine should set the symbol to be used for marking stationary point positions when a call to PLTMK is made. The correspondence between graphical symbols and integer codes is determined by the user's implementation of this routine. By convention negative integers should suppress plotting.

## 9) SUBROUTINE PLTMK(X,Y)

This routine should move the current plotter position to the point (X,Y) invisibly and print a symbol centred there. The symbol will have been specified by a previous call to PLTSYM.

## 10) SUBROUTINE PLTOFF

This subroutine should close the plot frame opened by PLTON and leave the graphics system ready for another call to PLTON to create a new frame or for being terminated prior to the STOP statement in the user's program.

## SUMMARY OF HIGH LEVEL SUBROUTINES

At present CONICON 2 contains eight master routines, each of which is capable of producing a complete contour plot incorporating a number of the features described earlier. These routines may be divided into four pairs of routines which are listed below; in each case the first routine of the pair is a simple interface routine which carries out the task of opening and closing of the plot frame (by calling subroutines PLTON and PLTOFF), plots the boundary of the grid in a style chosen by the user, and calls the other routine of the pair. The user should therefore call one of subroutines CONIC1, CONXA1, CONXB1 and CONGR1 if he wishes his plot frame to be defined automatically, and he should call CONIC2, CONXA2, CONXB2 or CONGR2 if he wishes to define the plot frame himself. Use of any of the latter four routines will enable the user to overplot two or more contour maps easily, or to relate the spatial locations of phenomena such as data sites, geographical features etc. to the variable being contoured. Subroutines CONIC1 and CONGR2 plot all contours in the current logical pen colour (as defined by the most recent call to PLTPEN), and this is unchanged on exit. All other master routines plot contours using solid lines. Besides the differences outlined above, the subroutines in each pairing carry out identical tasks. The names of these master routines and the features which they offer are as follows:-

| | |
|---|---|
| CONIC1, CONIC2 | (1), (2), (3) and (5) |
| CONXA1, CONXA2 | (1), (2), (3) and (4) |
| CONXB1, CONXB2 | (1), (2), (3) and (4) |
| CONGR1, CONGR2 | (1), (2), (3), (5) and (7) |

Of the routines which produce standard contour plots of a surface, the first pair i.e. CONIC1 and CONIC2 are the simplest to use and have the shortest argument lists. Their use is recommended for any plot which does not make use of features (4) or (7).

Subroutines CONXA1, CONXA2, CONXB1 and CONXB2 are intended primarily for use when the crosshatching feature is being used (although they do not enforce its use). As has been discussed above, the package incorporates two fundamentally different crosshatching algorithms; the one which we refer to as algorithm A is used by subroutines CONXA1 and CONXA2, and the other, algorithm B, by subroutines CONXB1 and CONXB2. Neither algorithm can be considered superior to the other on all occasions; a discussion of their relative merits was presented in a previous section.

Subroutines CONGR1 and CONGR2 produce contour plots of the gradient of the surface. These maps are intended to help the user to identify those contours which are relatively unreliable. The user is offered exactly the same options with these routines as he has when using CONIC1 or CONIC2; indeed the argument lists of CONGR1 and CONGR2 are identical to those of CONIC1 and CONIC2 respectively.

Besides these eight master routines for contour plotting, a number of other high level routines are documented below. These are subroutines STPLT1, STPLT2, AUTOXH, GRSET, GRSUB, CONVEX and BORDER.

STPLT1 and STPLT2 are master routines which may be called to mark the locations of stationary points of the approximant function; the former routine opens and shuts the plot frame and will produce a plot frame of exactly the same scale and location as subroutines CONIC1, CONXA1, CONXB1 and CONGR1 produce. Subroutine STPLT2 leaves the task of opening and closing of the plot frame to the user.

Subroutine AUTOXH takes much of the pain out of crosshatching in many cases; it may be used to set crosshatching parameters (for algorithm B) automatically in examples where the user wishes the intensity of hatching to increase progressively as the height of the surface increases. Full use is made of the opportunity for superposition of hatches offered by algorithm B, and results will normally be obtained with considerably less expense than comparable plots produced using algorithm A. Subroutine AUTOXH may be called prior to calling CONXB1 or CONXB2 but should not be used in combination with algorithm A.

Subroutine GRSET should be called prior to calling any of the master routines if gradient values have to be estimated, unless the user has opted to omit contouring within some grid cells. In this case subroutine GRSUB should be used instead.

Subroutine CONVEX automatically selects those cells of the grid which lie completely within a convex window specified by the user and flags the rest of the cells as dead. Subroutine BORDER may be used to plot the boundary of the contoured area which results from calling CONVEX.

Unless otherwise stated, variable types can be assumed to follow the standard convention. i.e. Variables beginning with the letters I, J, K, L, M or N are integer valued; all others are floating point variables. Single precision is used throughout.

If a variable name is prefixed by a pair of asterisks, this means that the value(s) of that variable must be set before the routine is called. If a variable name is prefixed by a single asterisk, then that variable may or may not have to be set before the routine is called, depending on which options have been chosen; which is the case should be obvious from reading the notes on that variable.

## MASTER ROUTINES

### SUBROUTINE CONXA1(Z,ZX,ZY,MM,M,N,ZLIM,GRID,CT,NCT,ICT,XY,NXY, CONT,K,KTOP,K3,NK3,II,ITH,ILAB,NFR,AV,UV,IV,JV,NC1,XI,NXI,XD,ABC, TG,LV,TS,NTR,ALAB,AN,NLAB,IPEN)

A master routine which produces a single contour plot incorporating any combination of features (1) - (4). Crosshatching is carried out using algorithm A; if it is not required, subroutine CONIC1 (which has a shorter argument list) should be used. This subroutine creates a complete plot frame starting with a call to PLTON and ending with a call to PLTOFF. The boundary of the map is plotted in a style selected by the user and contours are plotted using solid lines.

Explanation of arguments:-

** Z(MM,N)
An array of surface heights. Z(1,1) is the height in the SW corner. The first dimension increases as x increases, and the second dimension corresponds to the value of y.

** ZX(MM,N)
An array of partial derivatives in the x-direction.

** ZY(MM,N)
An array of partial derivatives in the y-direction.

** MM
The true first dimension of the arrays Z, ZX and ZY, and the second dimension of the array ZLIM.

** M
The first dimension of the arrays Z, ZX and ZY that is actually to be plotted. i.e. The no of columns in the grid. M must therefore be less than or equal to MM.

** N
No of rows in the grid and the second dimension of Z, ZX and ZY.

ZLIM(2,MM,N)
Working array.

** GRID
Distance between a pair of adjacent grid points.

* CT(NCT)
In the usual case (when ICT > 0), CT holds the levels at which contours are to be plotted. If ICT < 0, CT(1) and CT(2) hold lower and upper limits respectively between which all contours will be chosen to lie. If ICT = 0, all values in CT are ignored.

N.B. If crosshatching is required and contour levels are selected by the user, values in CT must be in strictly ascending order.

** NCT
The no of contours required and the length of CT.

** ICT
Indicates whether the user wishes contour levels to be chosen automatically or by himself.

ICT = 0 => completely automatic choice of contour levels. (The user must of course always specify the number of contours required).

ICT < 0 => semi-automatic choice of contour levels:- the user is required only to specify a pair of values (CT(1) and CT(2)) between which all NCT contours will be chosen, at regular intervals.

ICT > 0 => user specifies all contour levels himself.

XY(2,NXY)
Working array.

** NXY
This variable corresponds to the length of the second dimension of working arrays XY and CONT. The required value of NXY is highly data-dependent, but MAX0(500,50*M,50*N) should be sufficient in most cases. If the selected value of NXY turns out to be too small, the program will terminate at STOP 621 and will have to be re-run using a higher value of NXY.

CONT(2,NXY)

Working array.

K(3,KTOP)
Working array.

** KTOP
This variable corresponds to the length of the second dimension of the working array K. Like NXY it is data-dependent and should be set to approximately four fifths the value of NXY. If it is not large enough, the program will terminate at STOP 620.

K3(NK3)
Working array.

** NK3
This variable corresponds to the length of the working array NK3. It is data-dependent. A value of 100 should normally be large enough, but if the feature of local suppression of contour plotting is used then the user should double this figure. If NK3 turns out not to be large enough then the program will terminate at STOP 622.

** II
Specifies

(a) whether or not thick line contours are required.

(b) whether or not annotation (giving contour heights) is required.

(c) whether or not crosshatching is required.

To find the appropriate value of II, begin with II = 0.

Add 1 if thick line contours are required.
Add 2 if annotation is required.
Add 4 if crosshatching is required.

* ITH
Specifies positions of thick-line contours (if requested). If the user requires the ith contour to be the first one drawn with a thick line and j thin line contours between each pair of thick-line contours, ITH should be set equal to 10*i + j. ITH is ignored if thick line contours have not been requested - otherwise it is essential that i > 0 and j < 10.

* ILAB
Specifies which contours (if any) are to be annotated. Choose in exactly the same way as ITH is chosen.

* NFR
Specifies the number of decimal places to be included in each contour height label. If NFR > 4 or NFR < 0, a sensible choice will be made automatically for each label. NFR = 0 suppresses the decimal point. This variable is redundant if annotation is not requested.

* AV(NC1)
A vector specifying the angle from horizontal (in radians) at which each band of crosshatching is to be drawn. Users should note that values in this array will be slightly perturbed by CONICON to avoid numerical difficulties occuring when hatch lines are (almost) parallel with the seam lines of the piecewise quadratic. CONICON expects the values in AV to be reasonably simple submultiples of pi.

* UV(NC1)
A vector specifying the scale (measured in the same units as the x and y coordinates) of each level of crosshatching. The value selected corresponds to the perpendicular distance between a pair of adjacent lines when style 1 is used. Some experimentation may be necessary before suitable values are discovered, but a reasonable initial estimate would be to choose values in this vector to be of the order of one fiftieth of the length of an edge of the plot.

* IV(NC1)
A vector holding code values for styles of crosshatching used between each pair of contours. IV(1) holds the code number of the style of hatching to be used below the level of CT(1); IV(2) holds the code number for the style to be used between CT(1) and CT(2); and so on. IV is ignored if crosshatching has not been requested. N.B. In a map with NCT contours, NCT+1 (= NC1) styles of crosshatching are required. The code values are as follows:-

(1) Simple hatch
(2) Square crosshatch
(3) Rectangular crosshatch
(4) Long rectangular crosshatch
(5) Very long rectangular crosshatch
(6) Stepped rectangular crosshatch
(7) Stepped long rectangular crosshatch
(8) Stepped very long rectangular crosshatch
(9) Bonded crosshatch
(10) Divided square crosshatch
(11) Herringbone crosshatch
(12) Long herringbone crosshatch
(13) Square boxes
(14) Rectangular boxes

(15) Steps
(16) Square waves in phase
(17) Square waves out of phase
(18) Square labels
(19) Ladders
(20) Digit 0
(21) Digit 1
(22) Digit 2
(23) Digit 3
(24) Digit 4
(25) Digit 5
(26) Digit 6
(27) Digit 7
(28) Digit 8
(29) Digit 9
(30) Diamond crosshatch
(31) Triangular crosshatch
(32) Triangle and hexagon crosshatch
(33) Honeycomb
(34) Diamond boxes
(35) Triangular boxes
(36) Hexagonal boxes
(37) Triangular labels
(38) Indentures
(39) Six stars
(40) Plus signs
(41) Multiplication signs
(42) Eight stars
(43) Square crosshatch with one diagonal
(44) Square crosshatch with two diagonals
(45) Square crosshatch with alternate diagonals
(46) Interwoven crosshatch
(47) Basketwork
(48) Conifers
(49) Deciduous forest
(50) Mixed forest
(51) Orchard

> 51 or < 1   no crosshatching

* JV(NC1)
A vector specifying the logical pen colour to be used at each level of crosshatching.

** NC1
NC1 = NCT+1, and is the length of vectors AV, UV, IV JV.

XI
Working array of dimension (5,NXI).

** NXI
Corresponds to the length of the second dimension of the array XI, which is used in crosshatching. Therefore if crosshatching is not required NXI should be set equal to 1. Otherwise a value of 10+MAX0(M,N) will usually be sufficient. NXI must never exceed the value of NTR (q.v.). If, on the other hand, it is not set large enough the program will terminate at STOP 625.

XD, ABC
Working arrays, each of dimension (2,6,NTR)

TQ
Working array of dimension (2,2,NTR).

LV, TS
Working vectors of length NTR.

** NTR
The length of a number of working arrays required for crosshatching purposes. NTR should be set to 1 if crosshatching is not required, otherwise a value of MAX0(30,3*M*N/2) would be a reasonable initial estimate. If this is not sufficiently large the program will terminate at STOP 623. The value of NTR should be kept as low as possible since 30*NTR storage locations are reserved. On some systems it may not be possible to set NTR to a sufficiently large value to cope with large maps.

ALAB(5,NLAB)
Working array.

AN(3,NLAB)
Working array

** NLAB
This variable corresponds to the length of the second dimensions of the arrays ALAB and AN. These arrays are only used when both crosshatching and annotation have been specified (i.e. II = 6 or 7). Thus if II < 6, NLAB should be set to 1. Otherwise NLAB must be at least as great as the total number of contour labels which will occur in the plot. A value of 5*NCT will usually be sufficient :- if it is not the program will terminate at STOP 624.

** IPEN
The logical pen colour used in plotting the boundary of the grid. The styles available will be implementation dependent. As usual, a negative value suppresses plotting.

```
SUBROUTINE CONXA2(Z,ZX,ZY,MM,M,N,ZLIM,GRID,CT,NCT,ICT,XY,NXY,
CONT,K,KTOP,K3,NK3,II,ITH,ILAB,NFR,AV,UV,IV,JV,NC1,XI,NXI,XD,ABC,
TO,LV,TS,NTR,ALAB,AN,NLAB,XSW,YSW)
-----------------------------------------------------------------
```

A master routine which produces a single contour plot incorporating any combination of features (1) - (4). Crosshatching is carried out using algorithm A; if it is not required, subroutine CONIC2 (which has a shorter argument list) should be used. The user is expected to scale the plot himself (i.e. he must call PLTON and PLTOFF) and plotting of the edge of the grid is also left to the user. Contours are plotted using solid lines.
Those arguments which appear in the argument list of CONXA1 have the same meaning here. The two arguments not defined previously are as follows:-

** XSW,YSW
Cartesian coordinates of the most south-westerly node of the grid. i.e. the point at which the surface has height and gradients Z(1,1), ZX(1,1), ZY(1,1). The nodes of the grid are assumed to be aligned parallel with the coordinate axes.

```
SUBROUTINE CONXB1(Z,ZX,ZY,MM,M,N,ZLIM,GRID,CT,NCT,ICT,XY,NXY,
CONT,K,KTOP,K3,NK3,II,ITH,ILAB,NFR,XH,AV,OV,UV,IV,JV,NH,XI,NXI,
ALAB,AN,NLAB,IPEN)
---------------------------------------------------------------
```

A master routine which produces a single contour plot incorporating any combination of features (1) - (4). Crosshatching is carried out using algorithm B; it is not obligatory, but if it is not required, subroutine CONIC1 (which has a shorter argument list) should be used. This subroutine creates a complete plot frame starting with a call to PLTON and ending with a call to PLTOFF. The boundary of the map is plotted in a line style selected by the user, and contours are plotted with solid lines.

All those arguments which appear in the argument list of subroutine CONXA1 have the same meaning here, with the following exceptions:-

** AV,UV,IV,JV
These arrays are now each of length (at least) NH (see below); NC1 is not an argument of this subroutine.

Those arguments which do not appear in the argument list of subroutine CONXA1 have the following meanings:-

** XH(2,NH)
An array holding the contour levels which define the positions of the NH bands of crosshatching. The area covered by each band of crosshatching is defined by a pair of contour levels, say C1 and C2 (where C1 < C2). To set this as the Jth band of crosshatching the user must set XH(1,J) = C1 and XH(2,J) = C2. If it is desired to crosshatch the whole area below a certain level, say C, with the Jth band, then XH(2,J) should be set equal to C, and XH(1,J) should be set to a level below the minimum value of the surface (a value below -1.0E+25 will give maximum efficiency). Similarly, to hatch the area above level C with the Jth band of hatching, the user should set XH(1,J) equal to C and XH(2,J) to a value higher than the maximum height of the surface, and preferably greater than 1.0E+25.

* OV(NH)
A vector of values (measured in the same units as the x and y coordinates) specifying the offset of an arbitrary line in the raster from the origin (the bottom left hand corner of the plot if scaling is carried out automatically). Each value in this vector is ignored unless the corresponding value in the array IV is 1, indicating that the simple hatch style has been selected. Control of this variable is useful in examples where the user wishes to create the effect of progressive darkening of the surface as its height increases, by superposition of bands of style 1; in such examples the user can increase the density of the hatching bands in a regular manner in much smaller steps than would otherwise be possible.

** NH
The number of bands of crosshatching to be plotted, and the length of the arrays AV, UV, IV, JV and the second dimension of the array XH.

SUBROUTINE CONXB2(Z,ZX,ZY,MM,M,N,ZLIM,GRID,CT,NCT,ICT,XY,NXY,
CONT,K,KTOP,K3,NK3,II,ITH,ILAB,NFR,XH,AV,OV,UV,IV,JV,NH,XI,NXI,
ALAB,AN,NLAB,XSW,YSW)
-----------------------------------------------------------------

A master routine which produces a single contour plot incorporating any combination of features (1) - (4). Crosshatching is carried out using algorithm B; if it is not required, it is recommended that subroutine CONIC2 (which has a shorter argument list) is used instead. The user is expected to scale the plot himself (i.e. he must call PLTON and PLTOFF) and

plotting of the boundary of the plot is also left to the user. Contours are plotted using solid lines.
Those arguments which appear in CONXD1's argument list have the same meaning here, and arguments XSW, YSW are also as defined previously.

SUBROUTINE CONIC1(Z,ZX,ZY,MM,M,N,ZLIM,GRID,CT,NCT,ICT,XY,NXY,
CONT,K,KTOP,K3,NK3,II,ITH,ILAB,NFR,IPEN)

---

A master routine allowing all the options available in subroutine CONXA1 and CONXB1 with the exception of crosshatching. The feature of local contour suppression is also available. If crosshatching is not required this routine should be used in preference to CONXA1 or CONXD1, as its argument list is considerably shorter. This subroutine creates a complete plot frame starting with a call to PLTON and ending with a call to PLTOFF. The boundary of the grid (which may or may not be the boundary of the contoured area) is plotted in a line style selected by the user and contours are plotted using solid lines.

All arguments also appear in the argument list of CONXA1 and are described in the documentation for CONXA1 above; however the following variables are defined differently in this routine:-

* ZLIM(2,MM,N)
When II > 3, this array indicates which cells of the grid should be contoured and which ones should not. The values ZLIM(1,I,J) and ZLIM(2,I,J) refer to the grid cell with surface height Z(I,J) in its SW corner. If the user wishes to contour within this cell he should set ZLIM(1,I,J) less than or equal to ZLIM(2,I,J) (values of 0.0 in each will do). Otherwise, if he wishes to suppress contour plotting within the area of this cell, he should set ZLIM(1,I,J) greater than ZLIM(2,I,J). Note that it is not necessary to set values in this array for I > M-1 and J > N-1, since we are dealing with an (M-1) * (N-1) grid of cells.
When II < 4 (i.e. the complete grid is to be contoured) values in this array need not be set by the user.
Note that subroutine CONVEX (see below) sets these values automatically in a common special case.

* CT(NCT)
Values in this array are no longer required to be ordered.

** II
This variable should be chosen in the following way:-

Begin with II = 0.
Add 1 if thick-line contours are required.
Add 2 if annotation is required.
Add 8 if contouring is to be restricted to a subset of the grid cells.

## SUBROUTINE CONIC2(Z,ZX,ZY,MM,M,N,ZLIM,GRID,CT,NCT,ICT,XY,NXY, CONT,K,KTOP,K3,NK3,II,ITH,ILAB,NFR,XSW,YSW)

A master routine which carries out the same tasks as CONIC1, with the exception that the user is expected to scale the plot himself (i.e. he must call PLTON and PLTOFF) and plotting of the edge of the grid or the boundary of the contoured area is also left to the user. Contours are plotted in the current logical pen colour (as specified by the most recent call to PLTPEN), which remains unaltered on exit.
Those arguments which also appear in the argument list of subroutine CONIC1 have the same meaning here, and the other two arguments (XSW and YSW) are as defined above.

## SUBROUTINE CONGR1(Z,ZX,ZY,MM,M,N,ZLIM,GRID,CT,NCT,ICT,XY,NXY, CONT,K,KTOP,K3,NK3,II,ITH,ILAB,NFR,IPEN)

A master routine which produces a complete contour plot of the gradient of a surface. Features (1), (2), (3) and (5) are also available. This subroutine creates a complete plot frame starting with a call to PLTON and ending with a call to PLTOFF. The boundary of the grid is plotted in a style selected by the user and contours are plotted using solid lines.
The argument list of this routine duplicates that of CONIC1. All arguments have the same meaning as their counterparts in that routine.

## SUBROUTINE CONGR2(Z,ZX,ZY,MM,M,N,ZLIM,GRID,CT,NCT,ICT,XY,NXY, CONT,K,KTOP,K3,NK3,II,ITH,ILAB,NFR,XSW,YSW)

A master routine which produces a complete contour plot of the gradient of the surface. Features (1), (2), (3) and (5) are also available. The user is expected to scale the plot himself (i.e.

he must call PLTON and PLTOFF) and plotting of the boundary of the contoured area is also left to the user. Contours are plotted in the current logical pen colour (as specified by the most recent call to PLTPEN), which remains unaltered on exit.
The argument list of this routine duplicates that of CONIC2 and all arguments have the same meaning here.

## OTHER SUBROUTINES

### SUBROUTINE STPLT1(Z,ZX,ZY,MM,M,N,ZLIM,GRID,ISYM)

A master routine which calculates and plots stationary points (local maxima, local minima and saddle points) of the approximant function (NOT the true function being contoured).
This routine carries out the tasks of opening and closing of the plot frame in such a way as to be compatible with subroutines CONIC1, CONXA1, CONXB1 and CONGR1, but unlike these routines it does not plot the boundary of the grid.
All arguments which appear in the argument list of CONIC1 have the same meaning here, with the following exception:-

** ZLIM(2,MM,N)

If the user wishes to suppress the plotting of stationary values within some cells of the grid then this array indicates which cells of the grid should be ignored.
Values should be set in the same manner as they are set to suppress contouring within grid cells. i.e. if the user wishes to suppress plotting of stationary values within the (I,J)th cell (the cell with height Z(I,J) in its SW corner) he should set ZLIM(1,I,J) greater than ZLIM(2,I,J). Otherwise, if the user requires stationary points within the (I,J)th cell to be calculated he should set ZLIM(1,I,J) no greater than ZLIM(2,I,J).
If the user's call to STPLT1 was preceded by a call to any of CONICON'S master contour plotting routines and the user has not altered values in ZLIM then plotting of stationary points will take place within all those cells of the grid which have been contoured, and no others.

The extra argument is:-

** ISYM(3)
A vector specifying the symbols used for plotting of local minima, saddle points and local maxima respectively. If a value in this array is negative then plotting is suppressed for the corresponding type of stationary point. Symbols available will vary according to the implementation.

### SUBROUTINE STPLT2(Z,ZX,ZY,MM,M,N,ZLIM,GRID,XSW,YSW,ISYM)

A master routine which calculates and plots stationary points (local maxima, local minima and saddle points) of the approximant function (NOT the true function being contoured).
 The user is expected to scale the plot himself (i.e. he must call PLTON and PLTOFF) and plotting of the boundary of the grid is also left to the user.
 All arguments which appear in the argument list of STPLT1 have the same meaning here. The other arguments (XSW and YSW) are as defined above.

## SUBROUTINE AUTOXH(M,N,GRID,CT,NCT,XH,AV,OV,UV,IV,JV,NX,NH)

Sets values of crosshatching parameters XH, AV, OV, UV, IV, JV and NH (for use by algorithm B) automatically in examples where the user wishes the intensity of hatching to increase progressively as the height of the surface increases. Full use is made of the opportunity for superposition of hatches offered by algorithm B, and results will normally be achieved with considerably less expense than comparable plots produced using algorithm A. This routine should be used prior to calling either CONXB1 or CONXB2, but should not be used in conjunction with hatching algorithm A. The routine will produce a maximum number of twelve bands of crosshatching (including the area below the bottom contour which will not be hatched); if the number of contour levels NCT implies a larger number than this (i.e. NCT > 10) then some contiguous areas will be hatched in the same style. On return from this routine the first NH values in the array IV will be set to 1, with the exception of IV(1) which will be zero. The first NH values in JV will all be zero.
 As the number of bands of hatching NH is unknown at the outset, a separate variable NX is used for dimensioning the arrays whose values are set by this routine; thus XH is an array of dimension (2,NX) and AV, OV, UV, IV, and JV are all vectors of length NX. The variables CT and NCT have the same meaning as previously; values in CT must be in ascending order and must have been set by the user. This routine should NOT therefore be used if the user has opted for automatic selection of contour levels (i.e. if ICT is non zero). NX must always be at least twelve, or a hard error will occur. As long as NX satisfies this requirement the value of NH which is returned will be less than or equal to 12, and no dimensioning problems will occur when CONXB1 or CONXB2 are called with NH and not NX as an argument.
 If the scale of hatches produced by this routine turns out to be unsuitable for the particular graphics device being used then the user should simply rescale all values in OV and UV after a call to this routine by multiplying by the same constant.

## SUBROUTINE GRSET(Z,ZX,ZY,MM,M,N,GRID,JJ)

This subroutine can be used to estimate gradient values given surface heights stored in the array Z. It should only be called before contouring over a complete rectangular grid, or in other cases where every value in the array Z is the true surface height at the appropriate location (Subroutine GRSUB should be used for gradient estimation in all other cases). Estimation is done by fitting a parabola over a point and its two nearest neighbours in the relevant direction. All parameters except JJ occur in CONXA1 and have the same meaning here. JJ has the following meaning:-

** JJ
In the usual case JJ should be set to 0. However if the surface can be considered to be a surface over a cylinder, then JJ should be set to 1 or 2 : 1 if the two vertical edges of the grid are identical and 2 if the two horizontal edges are identical. If the surface can be considered to be a surface over a torus, so that both the vertical edges are identical and the horizontal edges are identical, JJ should be set equal to 3.

## SUBROUTINE GRSUB(Z,ZX,ZY,MM,M,N,ZLIM,GRID)

This subroutine can be used to estimate gradient values given surface heights stored in the array Z. It should only be used in examples in which the user has opted to suppress contouring within some cells of the grid, and he has therefore set values in the array ZLIM to indicate which cells are to be contoured. These values may have been set by subroutine CONVEX (see below) or directly by the user.
As in GRSET, estimation is done by fitting a parabola over a point and its two nearest neighbours in the relevant direction; however this is subject to the condition that gradient estimates must only be based on surface heights which lie within (or on the boundary of) the contoured area, and therefore it is sometimes only possible to estimate gradients by fitting a straight line through a point and the nearest neighbour lying within the contoured area.

The argument ZLIM has the same meaning here as it has when it occurs as an argument of CONIC2 (with II > 3). All other arguments have been defined uniquely above.

## SUBROUTINE CONVEX(ZLIM,MM,M,N,GRID,XSW,YSW,CN,JCNS)

Identifies those grid cells which lie entirely within a user-defined convex window and flags all other cells as dead by making appropriate entries in the array ZLIM. An M * N grid of values and gradients is defined, with the addition of a pair of coordinates to fix its location. The nodes of the grid are assumed to be aligned parallel with the coordinate axes. A number of straight line constraints are defined in addition to this grid, in order to specify the window, and those cells of the grid which fail to satisfy any of the constraints are flagged as dead.

Arguments which require explanation are as follows:-

ZLIM(2,MM,N)
Unset on entry. On return this is in the correct form required as input data for CONIC2.

** CN(3,JCNS)
An array of constraints which specify the window within which we wish to contour the surface. Each constraint has the form

$$a*x + b*y + c < 0$$

and to set this as the Jth constraint the user must set CN(1,J) to a, CN(2,J) to b, and CN(3,J) to c.

** JCNS
The number of constraints and also the second dimension of CN.

## SUBROUTINE BORDER(ZLIM,MM,M,N,GRID,XSW,YSW,IPEN)

Traces round the boundary of the area which subroutine CONVEX has specified for contouring. Subroutine BORDER must only be called after a call to CONVEX.

Arguments have the same meaning as previously. There is one argument which has not been defined elsewhere:-

** IPEN
The logical pen colour used in plotting. IPEN = 0 corresponds to solid lines. Other styles may be available, depending on the implementation.

# APPENDIX B

## INTEGRAL EVALUATIONS

In Chapter 2 we presented an expression (2.17) for the integral of the seamed quadratic element as a linear combination of the values and gradients at its vertices, and discovered that the piecewise cubic element introduced in the same chapter has an identical expression as its integral.

In addition to this, and as a preliminary to the execution of some work which had to be abandoned for reasons of limited time, we have also evaluated the integral of the square of both piecewise quadratic and piecewise cubic elements over their extents. Although we have not actually put these results to any practical use they may be of potential value to readers wishing to carry out further investigation of either of these elements, and we therefore present these results in this Appendix.

We begin though, by giving expressions for the integrated squares and cross-product of the one-dimensional analogues of our elements. Referring to the one-dimensional piecewise quadratic element as $f_q(x)$ and the cubic element as $f_c(x)$, we find

$$\int_{-h}^{h} f_q^2(x)\,dx = \frac{h}{35}(26\, z_L^2 + 18\, z_L z_R + 26\, z_R^2) + \frac{h^2}{105}(44\, z_L g_L - 26\, z_L g_R + 26\, z_R g_L - 44\, z_R g_R) + \frac{h^3}{105}(8\, g_L^2 - 12\, g_L g_R + 8\, g_R^2) \qquad (B.1)$$

$$\int_{-h}^{h} f_c^2(x)\,dx = \frac{h}{30}(23\, z_L^2 + 14\, z_L z_R + 23\, z_R^2) + \frac{h^2}{60}(27\, z_L g_L - 13\, z_L g_R + 13\, z_R g_L - 27\, z_R g_R) + \frac{h^3}{60}(5\, g_L^2 - 6\, g_L g_R + 5\, g_R^2) \qquad (B.2)$$

$$\int_{-h}^{h} f_q(x)\, f_c(x)\, dx = \frac{h}{240}(181\, z_L^2 + 118\, z_L z_R + 181\, z_R^2) + \frac{h^2}{30}(13\, z_L g_L - 7\, z_L g_R$$

$$+ 7\, z_R g_L - 13\, z_R g_R) + \frac{h^3}{240}(19\, g_L^2 - 26\, g_L g_R + 19\, g_R^2) \qquad (B.3)$$

It follows that the integrated squared difference between the surfaces is simply:

$$\int_{-h}^{h} (f_q(x) - f_c(x))^2\, dx = \frac{h}{240}\{z_L - z_R + h\,(g_L + g_R)\}^2 \qquad (B.4)$$

The expression in curly brackets is of course the difference between tangents constructed at the endpoints of the element, evaluated at its midpoint; we know that this is zero when the true underlying function is quadratic.

We move on now to look at the two dimensional elements, $f_q(x, y)$ and $f_c(x, y)$. Unfortunately we have not had time to evaluate the integral of their cross-product and consequently the integrated squared difference between the two approximant surfaces. Particularly in view of the result (B.4) above this would appear to be a worthwhile task.

$$\int_{y=-k}^{k} \int_{x=-h}^{h} f_q^2(x, y) = \frac{7}{12}(z^2_{SW} + z^2_{SE} + z^2_{NE} + z^2_{NW})\, hk$$

$$+ \frac{11}{30}(z_{SW}\, z_{SE} + z_{SW}\, z_{NW} + z_{SE}\, z_{NE} + z_{NW}\, z_{NE})\, hk$$

$$+ \frac{1}{10}(z_{SW}\, z_{NE} + z_{SE}\, z_{NW})\, hk$$

$$+ \frac{481}{1440}(z_{SW}\, s_{SW} - z_{SE}\, s_{SE} + z_{NW}\, s_{NW} - z_{NE}\, s_{NE})\, h^2 k$$

$$+ \frac{243}{1440}(-z_{SW}\, s_{SE} + z_{SE}\, s_{SW} - z_{NW}\, s_{NE} + z_{NE}\, s_{NW})\, h^2 k$$

$$+ \frac{167}{1440} (z_{SW} s_{NW} - z_{SE} s_{NE} + z_{NW} s_{SW} - z_{NE} s_{SE}) h^2 k$$

$$+ \frac{69}{1440} (-z_{SW} s_{NE} + z_{SE} s_{NW} - z_{NW} s_{SE} + z_{NE} s_{SW}) h^2 k$$

$$+ \frac{481}{1440} (z_{SW} t_{SW} + z_{SE} t_{SE} - z_{NW} t_{NW} - z_{NE} t_{NE}) hk^2$$

$$+ \frac{243}{1440} (-z_{SW} t_{NW} - z_{SE} t_{NE} + z_{NW} t_{SW} + z_{NE} t_{SE}) hk^2$$

$$+ \frac{167}{1440} (z_{SW} t_{SE} + z_{SE} t_{SW} - z_{NW} t_{NE} - z_{NE} t_{NW}) hk^2$$

$$+ \frac{69}{1440} (-z_{SW} t_{NE} - z_{SE} t_{NW} + z_{NW} t_{SE} + z_{NE} t_{SW}) hk^2$$

$$+ \frac{59}{960} (s^2_{SW} + s^2_{SE} + s^2_{NW} + s^2_{NE}) h^3 k$$

$$+ \frac{74}{960} (-s_{SW} s_{SE} - s_{NW} s_{NE}) h^3 k$$

$$+ \frac{42}{960} (s_{SW} s_{NW} + s_{SE} s_{NE}) h^3 k$$

$$+ \frac{22}{960} (-s_{SW} s_{NE} - s_{SE} s_{NW}) h^3 k$$

$$+ \frac{59}{960} ( t^2_{SW} + t^2_{SE} + t^2_{NW} + t^2_{NE}) hk^3$$

$$+ \frac{74}{960} (-t_{SW} t_{NW} - t_{SE} t_{NE}) hk^3$$

$$+ \frac{42}{960} (t_{SW} t_{SE} + t_{NW} t_{NE}) hk^3$$

$$+ \frac{22}{960} (-t_{SW} t_{NE} - t_{SE} t_{NW}) hk^3$$

$$+ \frac{137}{1440} (s_{SW} t_{SW} - s_{SE} t_{SE} - s_{NW} t_{NW} + s_{NE} t_{NE}) h^2 k^2$$

$$+ \frac{75}{1440} (s_{SW} t_{SE} - s_{SW} t_{NW} + s_{SE} t_{NE} - s_{SE} t_{SW} + s_{NW} t_{SW} - s_{NW} t_{NE}$$

$$+ s_{NE} t_{NW} - s_{NE} t_{SE}) h^2 k^2$$

$$+ \frac{33}{1440} (-s_{SW} t_{NE} + s_{SE} t_{NW} + s_{NW} t_{SE} - s_{NE} t_{SW}) h^2 k^2 \qquad \text{(B.5)}$$

$$\int_{y=-k}^{k} \int_{x=-h}^{h} f_c^{\,2}(x, y)\, dx\, dy = \frac{155}{280} (z^2_{SW} + z^2_{SE} + z^2_{NE} + z^2_{NW}) hk$$

$$+ \frac{106}{280} (z_{SW} z_{SE} + z_{SW} z_{NW} + z_{SE} z_{NE} + z_{NW} z_{NE}) hk$$

$$+ \frac{38}{280} (z_{SW} z_{NE} + z_{SE} z_{NW}) hk$$

$$+ \frac{261}{840} (z_{SW} s_{SW} - z_{SE} s_{SE} + z_{NW} s_{NW} - z_{NE} s_{NE}) h^2 k$$

$$+ \frac{155}{840} (-z_{SW} s_{SE} + z_{SE} s_{SW} - z_{NW} s_{NE} + z_{NE} s_{NW}) h^2 k$$

$$+ \frac{91}{840} (z_{SW} s_{NW} - z_{SE} s_{NE} + z_{NW} s_{SW} - z_{NE} s_{SE}) h^2 k$$

$$+ \frac{53}{840} (-z_{SW} s_{NE} + z_{SE} s_{NW} - z_{NW} s_{SE} + z_{NE} s_{SW}) h^2 k$$

$$+ \frac{261}{840} (z_{SW} t_{SW} + z_{SE} t_{SE} - z_{NW} t_{NW} - z_{NE} t_{NE}) hk^2$$

$$+ \frac{155}{840} (-z_{SW} t_{NW} - z_{SE} t_{NE} + z_{NW} t_{SW} + z_{NE} t_{SE}) hk^2$$

$$+ \frac{91}{840} (z_{SW} t_{SE} + z_{SE} t_{SW} - z_{NW} t_{NE} - z_{NE} t_{NW}) hk^2$$

$$+ \frac{53}{840} (-z_{SW} t_{NE} - z_{SE} t_{NW} + z_{NW} t_{SE} + z_{NE} t_{SW}) hk^2$$

$$+ \frac{287}{5040} (s^2_{SW} + s^2_{SE} + s^2_{NW} + s^2_{NE}) h^3k$$

$$+ \frac{434}{5040} (-s_{SW} s_{SE} - s_{NW} s_{NE}) h^3k$$

$$+ \frac{194}{5040} (s_{SW} s_{NW} + s_{SE} s_{NE}) h^3k$$

$$+ \frac{142}{5040} (-s_{SW} s_{NE} - s_{SE} s_{NW}) h^3k$$

$$+ \frac{287}{5040} (t^2_{SW} + t^2_{SE} + t^2_{NW} + t^2_{NE}) hk^3$$

$$+ \frac{434}{5040} (-t_{SW} t_{NW} - t_{SE} t_{NE}) hk^3$$

$$+ \frac{194}{5040} (t_{SW} t_{SE} + t_{NW} t_{NE}) hk^3$$

$$+ \frac{142}{5040} (-t_{SW} t_{NE} - t_{SE} t_{NW}) hk^3$$

$$+ \frac{55}{630} (s_{SW} t_{SW} - s_{SE} t_{SE} - s_{NW} t_{NW} + s_{NE} t_{NE}) h^2k^2$$

$$+ \frac{33}{630} (s_{SW} t_{SE} - s_{SW} t_{NW} + s_{SE} t_{NE} - s_{SE} t_{SW} + s_{NW} t_{SW} - s_{NW} t_{NE} + s_{NE} t_{NW} - s_{NE} t_{SE}) h^2k^2$$

$$+ \frac{19}{630} (-s_{SW} t_{NE} + s_{SE} t_{NW} + s_{NW} t_{SE} - s_{NE} t_{SW}) h^2k^2 \qquad (B.6)$$

The inelegant nature of these results is perhaps a little disappointing, but it is easy to verify that they are correct when the function being approximated is itself quadratic. If the analogy with the one dimensional elements still holds, then the expression for integrated squared difference should be very much simpler.