Bivariate splines

Stone, G.

Award date:
1988

Awarding institution:
University of Bath

Bivariate Splines.

submitted by G. Stone
for the degree of PhD
of the University of Bath
1988

Copyright

G. Stone

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Bivariate Splines.

Summary.

Cubic smoothing splines are well known to statisticians as the basis of a non-parametric regression technique with one independent and one dependent variable. They arise as the solution of the variational problem,

$$\underset{f}{\text{minimise}} \quad \sum_{i=1}^{N} (y_i - f(x_i))^2 + \alpha \int f''^2 \ ,$$

where $x_1 < \ldots < x_N$ are the design points, and $y_1, \ldots, y_N$ are the corresponding observations of the dependent variable, for some suitable choice of α, the smoothing parameter.

Their generalisation to two or more independent variables has also been widely studied, where the integral is generalised to the integral over $\mathbf{R}^n$ (for design points in n dimensions) of the sum of all the second (or higher) derivatives squared. However, the case where the region of integration is bounded is not so well explored. In this case the fitted splines are required to be smooth only over a region of interest containing the design points. An important special case of the above is given by bivariate spatial regression where the design points lie in the plane.

The solution of the variational problem in two dimensions corresponding to the univariate problem mentioned above, gives rise to a complicated boundary value problem, and the computation of an approximate solution to this problem is covered in this thesis. Chapter 1 presents the mathematical theory of the thin plate and the *finite window* splines, and is based upon the work of Wahba and of Dyn and Levin. We extend the work of Dyn and Levin to cover finite window smoothing splines.

In Chapter 2 we consider the implementations of both thin plate and finite window splines. We look at the efficient computation of a matrix required for both thin plate and finite window smoothing splines. We also develop a set of integration routines for use in the implementation of finite window splines, which are very much quicker than the numerical techniques proposed by Dyn and Levin. A discussion of the numerical ill-conditioning of the linear system involved in the thin plate splines is included and we look at one possible means of improving this conditioning. Chapter 3 is a comparison of the thin plate and finite window splines both in the context of interpolation with simulated data, and when smoothing a real data set.

An alternative approach, suitable for use on much larger data sets than either thin plate or finite window splines, results in the natural neighbour splines of R. Sibson. These splines arise from a discretisation of the roughness penalty, and give rise to a large sparse linear system. In Chapter 4 we consider the solution of this linear system using the method of conjugate gradients, which we describe, and investigate several preconditioning techniques. The resulting implementation is capable of smoothing a data set containing 10,000 points using a relatively small amount of processor time.

Chapter 5 is a discussion of an implementation of sparse Cholesky decomposition, for use with natural neighbour splines. We also compare the resulting splines to the finite window splines discussed earlier.

Acknowledgements

I would like to thank my supervisor, Prof. R. Sibson, for his guidance and assistance during the course of this work, and all the staff and postgraduates of the School of Mathematical Sciences, University of Bath. I acknowledge the financial support of the Science and Engineering Research Council.

Contents

Introduction.
Univariate Splines.

Consider the regression model,

$$y_i = f(t_i)+\varepsilon_i \quad i = 1,\ldots,N$$

where $t_1<\ldots<t_N\in\mathbf{R}$ are the design points, and $\{\varepsilon_i\}$ are assumed to be zero mean, uncorrelated errors, and f is some unknown function. We are interested in estimating f, given the observations y_i made at the design points. One way of obtaining an estimate is to assume that f belongs to some known family of curves parametrised by a relatively small number of parameters. The estimation of f then reduces to the estimation of the unknown parameters. This parametric approach has the problem that we must know or assume the family from which f is drawn. Nonparametric approaches to interpolation and smoothing avoid this problem, but at the cost of being less powerful in cases where an assumption of functional form is appropriate. One of the best known examples of such a method is cubic spline interpolation and smoothing.

Interpolating splines arise as the solution of a variational problem:

$$\begin{array}{rc} \text{minimise} & \int(f'')^2 \\ \text{subject to} & f\in H \\ \text{and} & f(t_i)=y_i \quad i=1,\ldots,N \end{array}$$

where here H, is some suitable space of functions from which f is chosen. This function space is not as restricted as a parametric family of curves, and in fact in our case we take H to be the set of functions whose (generalised) second derivatives are (Lebesgue) square integrable (see Chapter 1).

This problem arises in mechanics, as the problem of finding the shape of a stiff wire constrained to pass through certain points. Its relevance to data analysis is based on the property that the penalty function being minimised measures departure from a first degree function, and is in other words a *roughness penalty.*

It can be shown (see Ahlberg, Nilson and Walsh (1967) or de Boor (1978) for example) that the solution to this problem is unique, and the resulting function is a cubic polynomial between neighbouring data sites, with a continuous second derivative at the data sites. This solution is known as a piecewise cubic polynomial with knots at the data sites and a continuous second derivative. It is important to note that the functions in H do not all have continuous second derivatives, (only first derivatives), and that this continuity arises from the nature of the problem.

Univariate smoothing splines arise from the similar problem,

$$\begin{aligned} &\text{minimise} \quad E(f) + \alpha \int (f'')^2 \\ &\text{subject to} \quad f \in H \end{aligned}$$

where here E is some functional measuring the discrepancy between f and the data, and $\alpha > 0$ is a trade-off or smoothing parameter. For most purposes a suitable choice for E is (possibly weighted) sum of squared residuals, that is,

$$E(f) = \sum_{i=1}^{N} (y_i - f(t_i))^2 .$$

It can be shown that the solution of this problem is again a piecewise cubic polynomial with knots at the data sites, and has a continuous second derivative. The choice of smoothing parameter, which controls the trade-off between the roughness of the fitted function and its fidelity to the data, is the subject of much of the statistical literature concerning splines, (see for example Silverman (1985), Craven and Wahba (1979) and Hutchinson and De Hoog (1985)). We shall not consider this problem in this thesis.

It can be shown that the fitted values $\hat{y}_i = \hat{f}(t_i)$ (where $\hat{f}$ is the estimate of f) are related to the observations by,

$$\hat{Y} = A(\alpha)\, Y$$

where $A(\alpha)$ is an $N \times N$ matrix depending on α. This matrix is called the *hat matrix* and has some properties similar to those of the hat matrix in linear regression. (see Eubank (1984).)

Another property of the solution splines to the interpolation and smoothing problems is that they are first degree functions outside the range of the data. Notice that first degree functions are *free*, in the sense that $g'' = 0$ wherever g is first degree, so that there is no contribution to the roughness of the solution from outside the range of the data.

B-splines.

When fitting splines to a set of data the usual procedure is to define a basis for the space of all twice continuously differentiable piecewise cubic polynomials with knots at $t_1, \ldots, t_N$, which are linear outside (t_1, t_N), and then determine the coefficients of these basis functions in the representation of $\hat{f}$ by minimising the penalty. One such basis which arises naturally from one approach to the solution of the variational problem, is derived from the first degree functions together with the N Green's functions

$$|t-t_i|^3 \qquad (i=1,\ldots,N)$$

It can be shown that the solution spline to both the interpolation and smoothing problems can be expressed as,

$$\hat{f}(t) = a+bt+\sum_{i=1}^{N} c_i|t-t_i|^3$$

where a, b and the c_i are determined by the data y_i and α, and the c_i satisfy the extra conditions, to ensure first degree tail-out,

$$\sum c_i = \sum c_i t_i = 0.$$

However the determination of these coefficients involves the inversion of a matrix which although almost triangular, can be quite ill-conditioned (Eubank 1988).

A change of basis can be made to obtain a set of spline functions $\beta_1,\ldots,\beta_N$ with the following properties.

$$\begin{aligned} & \beta_i(t_i)>0 \\ \text{and} \quad & \beta_i(t)=0 \text{ if } t\notin(t_{i-2},t_{i+2}) \ (i=3,\ldots,N-2) \\ & \beta_i(t)=0 \text{ if } t\geq t_{i+2} \ (i=1,2) \\ & \beta_i(t)=0 \text{ if } t\leq t_{i-2} \ (i=N-1,N) \end{aligned}$$

This basis is called the B-spline basis, and is discussed further in Silverman (1985), Eubank (1988) and de Boor (1978).

The bounded support of the functions in the B-spline basis (when β_i is not an *edge* spline), described above, is very useful. This property means that all the matrices involved in the minimisation functional, which is now a quadratic form in the coefficients of the B-splines, are banded. Consequently quite large univariate data sets can be smoothed using smoothing splines, with a B-spline basis.

In the next section we discuss the generalisation of univariate splines with particular emphasis on bivariate splines. Further information can be obtained on other aspects of univariate splines from Silverman (1985) and Wegman and Wright (1983).

Bivariate splines.

There are several generalisations of splines from one to several dimensions. Perhaps the simplest is the concept of tensor product splines (de Boor (1978) and Hu and Schumaker (1986) for example). This method is applicable only to rectangular gridded data. In two dimensions this corresponds to defining two sets of B-splines, based on the knot sequences in the two directions in the grid. That is for a grid (x_i,y_j) with $1\leq i\leq n$ and $1\leq j\leq m$, define the B-splines $\{N_i\}$ on the

knot sequence $\{x_i\}$, and the B-splines $\{M_i\}$ on the knot sequence $\{y_i\}$. The spline surface is then given by,

$$s(x,y) = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}N_i(x)M_j(y)$$

where c_{ij} are coefficients determined by interpolation or smoothing criteria.

These splines have preferred directions, since the resulting surface is a piecewise cubic along any line parallel to one of the grid directions, but piecewise sextic along lines in other directions. The special form of the data site positions means also that these splines are inapplicable to most statistical applications.

Another possible generalisation of univariate splines is given by multivariate B-splines (Dahmen (1980,1981)). These functions are piecewise polynomials with compact support, and so have several nice properties similar to univariate B-splines. However, they rely on a triangulation of the data sites for their definition, and where there are ambiguities in the triangulation, for instance when a subset of the points lie in a cyclic configuration, it is not clear that the fitted surface is independent of the arbitrary choices made.

A third generalisation, (and the one which concerns us here), arises from a generalisation of the variational characterisation of univariate splines. We are interested only in bivariate splines here, so we only consider the generalisation to functions in the plane. The integral of the second derivative squared is generalised to the integral (over a suitable region) of the sum of all the second derivatives squared (including $\partial^2/\partial x\partial y$ and $\partial^2/\partial y\partial x$). This penalty function has the property that it is invariant under Euclidean transformations (rotation, reflection and translation). This gives rise to a characterisation discussed in Chapter 1. The solution of this characterisation inherits the above mentioned property of the penalty, and also is invariant under and isotropic change of scale. A further generalisation can be made by allowing the order of the derivatives taken to vary but we do not consider this here.

Remember that univariate splines are first degree outside the range of the data. This means that the univariate integral can be taken over any interval that contains all the data, and the same solution spline will result. In two dimensions however this is not, in general, the case. By a modification of the methods used in Chapter 2, Section 2 it is possible to show that bivariate splines are not first degree outside the range of the data, nor do they tend to a first degree function a long way from the data. Thus in theory the region of integration in the bivariate measure of roughness (see Chapter 1) can affect the solution spline. When this region is the whole plane, the solution is the so called thin plate spline, studied

extensively by Wahba (1979) etc. and others. The spline is known as a thin plate spline because the roughness in this case is proportional to the strain energy in a uniform thin plate of infinite extent, under infinitesimal displacements at the data sites. This problem in mechanics generalises the mechanical problem from which univariate splines arise. Dyn and Levin (1982) have suggested a numerical technique, when the region of integration is polygonal, in the interpolation case. In Chapter 1 we present the mathematical derivation of the solution splines in both the thin plate and what we have called the finite window cases, and extend the work of Dyn and Levin to cover smoothing splines. Chapter 2 describes several improvements to the existing implementations of bivariate spline procedures, and Chapter 3 is a comparison of thin plate and finite window splines.

There is no basis composed of functions with compact support for the bivariate problem as considered here, and all the matrices in the calculation of the splines are full and can be quite ill-conditioned. Section 2 of Chapter 2 considers this problem and suggests a change of basis which can reduce the condition numbers of the matrices involved. For larger problems (that is those involving large numbers of data sites) the denseness of the matrices becomes a limiting constraint. Sibson (1985) has proposed a discrete approximation to the roughness penalty used in finite window bivariate splines. This approximation gives rise to very sparse linear systems, and Chapter 4 describes this approximation and considers a class of sparse matrix techniques for the solution of the linear systems. Chapter 5 considers a direct method for solving the linear system and compares the resulting splines to the finite window splines that they approximate.

Chapter 1.
Mathematical Determination of Bivariate Splines.
1.1. Characterisation of Bivariate Interpolatory Splines.

This section summarises the mathematical aspects of spline theory that we require to find a bivariate interpolatory spline. We will consider smoothing splines later. The multivariate interpolatory spline problem can be stated,

$$\begin{aligned} &\underset{f\in H}{\text{minimise}} \quad J(f) \\ &\text{subject to} \quad f(t_i)=z_i \quad \text{for} \quad 1\le i\le N\ , \end{aligned}$$

where here $H\subseteq\{f:\mathbf{R}^n\to\mathbf{R}\}$ is a set of functions, with $J:H\to\mathbf{R}$ some measure of the *roughness* of a function in H and the t_i are points in a domain $\Omega\subseteq\mathbf{R}^n$, with $z_i\in\mathbf{R}$ the values to be interpolated.

Before we can solve this problem we must specify what we mean by J and which functions are to be included in the space H. A suitable class of functionals J which have been considered in detail are,

$$J_n(f)=\sum_{i_1,\ldots,i_m=1}^{n}\int_{\Omega\subseteq\mathbf{R}^n}\left(\frac{\partial^m f(x_1,\ldots,x_n)}{\partial x_{i_1}\ldots\partial x_{i_m}}\right)^2 dx_1\ldots dx_n$$

where $m\in\mathbf{N}\cup\{0\}$.

Our main interest will be confined to the case when $(n=2)$ so that we shall be surface fitting. We take $m=2$, the *natural* analogue of univariate cubic splines, so that, $\Omega\equiv\mathbf{R}^2$ corresponds to the well known *thin plate* spline problem. A more general discussion (that is with $m,n\neq 2$) of multivariate splines is given by Duchon (1976), Meinguet (1979a) and Meinguet (1979b). A discussion of the related smoothing splines can be found in Wahba (1979) and the case $(\Omega\neq\mathbf{R}^2)$ is considered by Dyn and Levin (1982).

In the area of our interest, however, J corresponds to the functional

$$J_\Omega(f)=\int_\Omega\left(\frac{\partial^2 f}{\partial x^2}\right)^2+2\left(\frac{\partial^2 f}{\partial x\partial y}\right)^2+\left(\frac{\partial^2 f}{\partial y^2}\right)^2$$

and here $J_{\mathbf{R}^2}(f)$ is the strain energy of a uniform thin plate of infinite extent under infinitesimal displacement f from the rest position.

It remains to choose the space of functions H, over which to minimise J_Ω. It is appealing to take

$$H=\left\{f\ \middle|\ \frac{\partial f}{\partial x},\frac{\partial f}{\partial y}\ \text{continuously differentiable},\ \frac{\partial^2 f}{\partial x^2},\frac{\partial^2 f}{\partial x\partial y},\frac{\partial^2 f}{\partial y^2}\in L^2(\Omega)\right\}.$$

However we will see later that we require our space H to be a Hilbert space under a norm derived from J_Ω. In order to have completeness we must include functions where $\partial f/\partial x$ and $\partial f/\partial y$ are not necessarily continuously differentiable, but which in some sense have discontinuous derivatives. To construct a suitable space, therefore, we make use of the notion of generalised functions (or distributions) and their generalised (or distributional) derivatives. For a more detailed treatment see Mazja (1980) and Temple (1955).

In this theory continuous functions have generalised derivatives of all orders. (In fact generalised functions have generalised derivatives of all orders.) We shall write $\partial_x f$ for the generalised derivative of f w.r.t. x. It can be shown that if $\partial f/\partial x$ is continuous in some domain then $\partial_x f$ is equal almost everywhere to an ordinary continuous function and $\partial_x f \equiv \partial f/\partial x$ in that domain.

Accordingly we define H to be $\{f : \partial_{xx} f, \partial_{xy} f, \partial_{yy} f \in L^2(\Omega)\}$. This space is known as the generalised Beppo-Levi space of order 2 over Ω. (See Deny and Lions (1954) and Meinguet (1979b)).

It can be shown, (Duchon (1976), Meinguet (1979a) and Mazja (1980)), that H, as defined above, equipped with the semi-inner product

$$A_\Omega(u,v) = \int_\Omega \partial_{xx} u \partial_{xx} v + 2\partial_{xy} u \partial_{xy} v + \partial_{yy} u \partial_{yy} v,$$

is a semi-Hilbert space of *continuous* functions.

Before proceeding further we make the following remark. Since throughout we are using Lebesgue measure any function in H is determined only up to addition of a function which is zero almost everywhere, so that when we say $u=v$ in H we mean $u=v$ *a.e.* and by v is continuous in H we mean $\exists\, w \equiv 0$ *a.e. s.t.* $v+w$ is continuous in the accepted sense.

The semi-inner product A_Ω as defined above has kernel on H,

$$\Phi = \text{span}\ \{\, 1\,, x\,, y\,\},$$

that is, the space of first-degree polynomials in x and y, (where we note the above proviso). Let us now return to our interpolation problem which can now be stated,

$$\underset{f \in H}{\text{minimise}}\ A_\Omega(f,f) \text{ s.t. } f(t_i) = z_i \quad (1 \le i \le N).$$

Suppose, without loss of generality, that t_{N-2}, t_{N-1}, t_N are not collinear. Then we can choose a basis p_1, p_2, p_3 for Φ such that, $p_i(t_{N-3+j}) = \delta_{ij}$ (the Kronecker delta) $1 \le i,j \le 3$, that is

		$p_i(t_{N-2})$	$p_i(t_{N-1})$	$p_i(t_N)$
	1	1	0	0
i	2	0	1	0
	3	0	0	1

We now define the linear map $P : H \to \Phi$ by

$$P(f) = f(t_{N-2})p_1 + f(t_{N-1})p_2 + f(t_N)p_3,$$

so that $P(f)$ is the first-degree polynomial taking the same values as f at t_{N-2}, t_{N-1} and t_N. We can see that P is a projection onto Φ with kernel given by

$$X_0 = (I-P)H = \{ f \in H \mid f(t_{N-2}) = f(t_{N-1}) = f(t_N) = 0 \}.$$

It can be seen that since H is a semi-Hilbert space then, equipped with A_Ω, X_0 is a Hilbert space and

$$H = \Phi \oplus X_0.$$

Note that the preceding discussion holds for any three distinct $t_i \in \{t_1, \ldots, t_N\}$ that are affine independent (ie. not collinear), or more generally we can write

$$X_0 \cong H/\Phi$$

(that is X_0 is isomorphic to the quotient space H/Φ).

So we may again restate our original problem in the following way.

Problem. *Let $f \in H$ be such that $f(t_i) = z_i$ for $1 \le i \le N$, then find $u^* \in (I-P_0)H$ with, $u^*(t_i) = (I-P_0)(f)(t_i)$ for $1 \le i \le N$, where P_0 is any linear projector from H onto Φ and such that $A_\Omega(u^*, u^*)$ is a minimum.*

In particular using the P defined above and writing,

$$z_i' = z_i - z_{N-2}p_1(t_i) - z_{N-1}p_2(t_i) - z_N p_3(t_i) \text{ for } 1 \le i \le N-3$$

we have the equivalent problem,

Problem. *Find $u^* \in X_0$ such that $u^*(t_i) = z_i'$ $1 \le i \le N-3$ and $A_\Omega(u^*, u^*)$ is a minimum.*

Put

$$\begin{aligned} W_0 &= \{ f \in X_0 \mid f(t_i) = 0 \quad 1 \le i \le N-3 \} \\ &\cong \{ f \in H \mid f(t_i) = 0 \quad 1 \le i \le N \}. \end{aligned}$$

Then W_0 is a *closed* subspace of X_0, since X_0 is a complete space of continuous functions. So the linear variety W defined by

$$W = \{ f \in X_0 \mid f(t_i) = z_i' \quad 1 \le i \le N-3 \}$$

is accordingly a closed affine subspace of X_0. In this situation we can use the orthogonal projection theorem to get,

Lemma. $\exists$ *unique* $u^* \in W$ *with minimal norm characterised by*

$$A_\Omega(u^*, v) = 0 \quad \forall v \in W_0$$

The orthogonal projection theorem is simply a generalisation of the geometric result that the closest approach of a line to a point $\mathbf{x}$ in $\mathbf{R}^2$ is achieved by the unique point $\mathbf{y}$ on the line where $(\mathbf{y}-\mathbf{x})\cdot\mathbf{z} = 0 \ \forall \mathbf{z}$ parallel to the line. In our case u^* is the point in W closest to the origin in X_0 and W_0 is the subspace parallel to W.

Since $A_\Omega(v,p) = 0 \ \forall p \in \Phi$ the above characterisation can be written.

Theorem 1.1. *There exists a unique solution* $\hat{u}$ *to the interpolation problem,*

$$\hat{u}(t_i) = z_i \quad 1 \le i \le N$$
$$\textit{with} \quad A_\Omega(\hat{u}, \hat{u}) \quad \text{a minimum}$$

and $\hat{u}$ *is characterised by*

$$A_\Omega(\hat{u}, v) = 0 \quad \forall v \in H \textit{ with } v(t_i) = 0 \quad 1 \le i \le N.$$

1.2. Identifying the Interpolatory Spline $\hat{u}$.

We know proceed to calculate $\hat{u}$, using the characterisation derived above and the existence and uniqueness properties provided by the orthogonal projection theorem.

Let $\beta_i \in H$ for $1 \le i \le N$ be such that $\beta_i(t_j) = \delta_{ij}$, $1 \le i, j \le N$. Such β_i exist since they could be simply interpolatory splines and these exist because of theorem 1.1. Then $\forall f \in H$,

$$f - \sum_{i=1}^{N} f(t_i)\, \beta_i \in W_0$$

and conversely any $g \in W_0$ can be written in this form, with $f \equiv g$, because $g(t_i) = 0$, $1 \le i \le N$ for all such g. So $\hat{u}$ is characterised completely by,

$$A_\Omega(\hat{u}, f - \sum_{i=1}^{N} f(t_i)\beta_i) = 0 \quad \forall f \in H \quad \text{and} \quad \hat{u} \in W.$$

That is $\hat{u}$ is completely characterised by,

$$A_\Omega(\hat{u}, f) = \sum_{i=1}^{N} A_\Omega(\hat{u}, \beta_i) f(t_i) \quad \forall f \in H$$

and the interpolation conditions.

The exact choice of the β_i in the above is not important, since, suppose that $\gamma_i \in H$, $1 \le i \le N$ are also such that $\gamma_i(t_j) = \delta_{ij}$ for $1 \le i,j \le N$. Then for each j since $\gamma_j \in H$,

$$\begin{aligned} A_\Omega(\hat{u},\gamma_j) &= \sum_{i=1}^{N} A_\Omega(\hat{u},\beta_i)\,\gamma_j(t_i) \\ &= A_\Omega(\hat{u},\beta_j) \end{aligned}$$

So $A_\Omega(\hat{u},\beta_{i)}$ is independent of the particular choice of β_i and we may write $\hat{u}$ is characterised by

$$A_\Omega(\hat{u},f) = \sum_{i=1}^{N} \lambda_i(\hat{u})\, f(t_i)$$

where $\lambda_i(\hat{u}) = A_\Omega(\hat{u},\beta_i)$ and β_i is *any* element of H satisfying $\beta_i(t_j) = 0$ for $j \neq i$ and $\beta_i(t_i) = 1$.

At this point we note that since $A_\Omega(\hat{u},p) = 0$ for all $p \in \Phi$ we must have,

$$\sum_{i=1}^{N} \lambda_i(\hat{u})\, p(t_i) = 0 \qquad \forall p \in \Phi \ .$$

We note that since $\hat{u}$ is the unique solution of the interpolatory spline problem, by theorem 1.1, the $\lambda_i(\hat{u})$ are uniquely determined by the interpolation conditions and the above.

This characterisation is sufficient in itself to determine the solution spline $\hat{u}$ up to a first-degree polynomial, for we can see that if,

$$\begin{aligned} & A_\Omega(u,f) = A_\Omega(v,f) && \forall\ f \in H, \\ \text{then} \quad & A_\Omega(u-v,f) = 0 && \forall\ f \in H, \\ \text{so that in particular} \quad & A_\Omega(u-v,u-v) = 0, \\ \Rightarrow\ & \partial_{xx}(u-v),\ \partial_{xy}(u-v) \text{ and } \partial_{yy}(u-v) = 0 \text{ in } \Omega, \\ \Rightarrow\ & (u-v) \in \Phi. \end{aligned}$$

We now proceed to the calculation of $\hat{u}$ from the above characterisation. In appendix A, we show the derivation of a Green's formula. This Green's formula is given in Dyn and Levin (1982) and below. We have for bounded Ω (provided $\partial\Omega$ satisfies certain regularity conditions),

$$\begin{aligned} A_\Omega(u,v) &= \int_\Omega \partial_{xx}u\partial_{xx}v + 2\partial_{xy}u\partial_{xy}v + \partial_{yy}u\partial_{yy}v \\ &= \int_\Omega (\nabla^4 u)\, v + \oint_{\partial\Omega} \left(\nabla\left(\frac{\partial u}{\partial n}\right)\cdot\nabla v - \frac{\partial}{\partial n}(\nabla^2 u)v\right) \end{aligned}$$

If Ω is not bounded this formula cannot hold as $\partial\Omega$ makes little sense. However following Meinguet (1979b) we can derive the equivalent formula for $\Omega \equiv \mathbf{R}^2$, namely,

$$A_{\mathbf{R}^2}(u,v) = \int_{\mathbf{R}^2} (\nabla^4 u) v \qquad \forall u,v \in H.$$

Thus we can derive an equivalent differential characterisation of $\hat{u}$, in terms of the Dirac delta function centred at a point t, δ_t. That is we can write

$$\begin{aligned} \nabla^4 \hat{u} &= \sum_{i=1}^{N} \lambda_i(\hat{u})\, \delta_{t_i} \quad \text{in } \Omega, \\ \nabla \frac{\partial \hat{u}}{\partial n} &= \mathbf{0} \qquad\qquad \text{on } \partial\Omega, \\ \frac{\partial}{\partial n} \nabla^2 \hat{u} &= 0 \qquad\qquad \text{on } \partial\Omega. \end{aligned}$$

where the boundary conditions are automatically satisfied when $\Omega \equiv \mathbf{R}^2$ (since $\hat{u} \in H$). Accordingly we look for $\hat{u}$ in the form,

$$\hat{u}(t) = \sum_{i=1}^{N} \lambda_i(\hat{u})\, E_\Omega(t,t_i) + p(t)$$

where $p \in \Phi$, and E_Ω is a solution to the following,

$$\nabla^4 E_\Omega(\cdot,t) = \delta_t \quad \text{in } \Omega \ ,$$

$$\left.\begin{aligned} \nabla \frac{\partial E_\Omega(\cdot,t)}{\partial n} &= \mathbf{0} \\ \frac{\partial}{\partial n} \nabla^2 E_\Omega(\cdot,t) &= 0 \end{aligned}\right\} \quad \text{on } \partial\Omega \ .$$

1.2.1. $\Omega \equiv \mathbf{R}^2$.

Firstly we shall consider the case when $\Omega \equiv \mathbf{R}^2$. The above equation is known in this case to have a solution given by

$$E(s,t) = (16\pi)^{-1} \|s-t\|^2 \log \|s-t\|^2 \qquad s,t \in \mathbf{R}^2$$

(See Duchon (1976), Dyn and Levin (1982), Meinguet (1979a) or Meinguet (1979b) for example. Note that these authors all write E in the form $(8\pi)^{-1} \|s-t\|^2 \log \|s-t\|$ but we use the above form so as to save the computation of a square root in the implementation.)

Unfortunately it can be seen that $E(\cdot,t) \notin H$ since $\partial_{xx} E(\cdot,t)$ and its other second derivatives are not square integrable functions. Meinguet (1979b) remarks that $E(\cdot,t)$ has square integrable second derivatives over any compact region, even if that region contains the point t and goes on to prove that the function

$$\sum_{i=1}^{N} \omega_i E(\cdot, t_i)$$

has square integrable second derivatives for all distinct $t_i \in \mathbf{R}^2$ and for any set $\{\omega_i : 1 \le i \le N\}$ provided,

$$\sum_{i=1}^{N} \omega_i = 0, \sum_{i=1}^{N} \omega_i x_i = 0, \sum_{i=1}^{N} \omega_i y_i = 0.$$

where $t_i = (x_i, y_i)$ $1 \le i \le N$.

Thus we have that,

$$A_{\mathbf{R}^2}(\sum_{i=1}^{N} \omega_i E(\cdot, t_i), f) = \sum_{i=1}^{N} \omega_i f(t_i),$$

for all t_i and ω_i that satisfy the above. Thus $\hat{u}$ is given by

$$\hat{u}(t) = \sum_{i=1}^{N} \lambda_i E(t, t_i) + d_1 p_1(t) + d_2 p_2(t) + d_3 p_3(t)$$

where the coefficients λ_i $1 \le i \le N$ and d_j $j = 1,2,3$ are given by the interpolation conditions,

$$\sum_{i=1}^{N} \lambda_i E((x_j, y_j), t_i) + d_1 + d_2 x_j + d_3 y_j = z_j \quad 1 \le j \le N$$

and by the conditions,

$$\sum_{i=1}^{N} \lambda_i = 0, \quad \sum_{i=1}^{N} \lambda_i x_i = 0, \quad \sum_{i=1}^{N} \lambda_i y_i = 0 .$$

In matrix form writing,

$$(\underset{N\times N}{K})_{ij} = E(t_i, t_j) , \qquad \underset{N\times 3}{T} = \begin{bmatrix} 1 & x_1 & y_1 \\ \vdots & \vdots & \vdots \\ 1 & x_N & y_N \end{bmatrix},$$

and $\underset{N\times 1}{\lambda} = \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_N \end{bmatrix}$, $\underset{3\times 1}{d} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$, $\underset{N\times 1}{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_N \end{bmatrix}$.

we have the matrix equations,

$$K\lambda + Td = z, \qquad T^{\mathrm{T}} \lambda = \mathbf{0}.$$

These equations can be written in a from which makes nature of the linear system involved more apparent, namely,

$$\begin{matrix} \\ N \\ 3 \end{matrix}\overset{\begin{matrix} <N> & <3> \end{matrix}}{\begin{bmatrix} K & T \\ T^{\mathrm{T}} & 0 \end{bmatrix}} \begin{matrix} \\ N \\ 3 \end{matrix}\begin{bmatrix} \lambda \\ d \end{bmatrix} = \begin{matrix} \\ N \\ 3 \end{matrix}\begin{bmatrix} z \\ 0 \end{bmatrix}.$$

We solve these equations using the method of Wahba (1979) with some modifications. Firstly it is noted that if $u_1, \ldots, u_{N-3}$ are $N-3$ linearly independent N-vectors such that each is also orthogonal to the columns of T, then $T^{\mathrm{T}}\lambda = \mathbf{0}$ ensures that λ is in the space spanned by the u's. In matrix notation, writing,

$$\underset{N\times N-3}{U} = (u_1, \ldots, u_{N-3})$$

then we have

$$\lambda = U\gamma$$

where γ is an $(N-3)$-vector. In fact for the case of smoothing we require, for Wahba's method, that the columns of U are orthonormal, that is, $U^{\mathrm{T}}U = I_{N-3}$.

Substituting this in the interpolation equation gives us,

$$KU\gamma + Td = z$$

so premultiplying this equation by U^{T} we have, using the fact that $U^{\mathrm{T}}T = \mathbf{0}$,

$$\underset{N-3\times N-3}{(U^{\mathrm{T}}KU)}\ \underset{N-3\times 1}{\gamma} = \underset{N-3\times 1}{(U^{\mathrm{T}}z)}$$

From this equation γ can be obtained, and hence λ. d is found from the equation,

$$\underset{3\times 3}{(T^{\mathrm{T}}T)}\ \underset{3\times 1}{d} = \underset{3\times N}{T^{\mathrm{T}}}\ (\underset{N\times 1}{z} - \underset{N\times N}{K}\ \underset{N\times 1}{\lambda})$$

and then $\hat{u}$ is completely determined.

1.2.2. $\Omega \neq \mathbf{R}^2$.

We now return to the problem of finding E_Ω when $\Omega \neq \mathbf{R}^2$. It is instructive to arrange the first two of the boundary conditions in an alternative form namely, reversing the order of differentiation gives,

$$\frac{\partial}{\partial n}\left[\frac{\partial E_\Omega(\cdot,t)}{\partial x}\right] = 0$$

and

$$\frac{\partial}{\partial n}\left[\frac{\partial E_\Omega(\cdot,t)}{\partial y}\right] = 0$$

on the boundary $\partial\Omega$. This says that the change in gradient across the boundary is zero and the remaining boundary condition says that the change in the Laplacian is zero across the boundary. To determine E_Ω we put $E_\Omega = E_1 + E_2$ with E_1, E_2

the solutions of the following, (see Dyn and Levin (1982))

$$\nabla^4 E_1(\cdot,t) = \delta_t \qquad \text{in } \Omega \ ,$$

$$\nabla^4 E_2(\cdot,t) = 0 \qquad \text{in } \Omega \ ,$$

$$\left.\begin{aligned} \nabla \frac{\partial E_2(\cdot,t)}{\partial n} &= -\nabla \frac{\partial E_1(\cdot,t)}{\partial n} \\ \frac{\partial}{\partial n} \nabla^2 E_2(\cdot,t) &= -\frac{\partial}{\partial n} \nabla^2 E_1(\cdot,t) \end{aligned}\right\} \qquad \text{on } \partial\Omega \ .$$

The solution for E_1 is easily found as in the case when $\Omega \equiv \mathbf{R}^2$. Thus we let,

$$E_1(s,t) = (16\pi)^{-1} \|s-t\|^2 \log\|s-t\|^2 \qquad s,t \in \Omega$$

To find E_2 however now entails the solution of a rather complicated boundary value problem which cannot in general be solved in closed form except in the case of some rather special choices of the region Ω. Dyn and Levin (1982) propose finding E_2 as a series solution.

They define a sequence of functions $\{\varphi_n\}$ by

$$\left.\begin{aligned} \varphi_1 &= \mathrm{Re}(\bar{z}z) \\ \varphi_{2+4j} &= \mathrm{Im}(z^{j+2}) \\ \varphi_{3+4j} &= \mathrm{Re}(z^{j+2}) \\ \varphi_{4+4j} &= \mathrm{Im}(\bar{z}z^{j+2}) \\ \varphi_{5+4j} &= \mathrm{Re}(\bar{z}z^{j+2}) \end{aligned}\right\} \qquad j \geq 0 \ .$$

where z is introduced as a short-hand notation, with $z = x+iy$.

Each member of this sequence satisfies the biharmonic equation, $\nabla^4 \varphi = 0$ in $\mathbf{R}^2$. In polar coordinates this sequence becomes,

$$\left.\begin{aligned} \varphi_1 &= r^2 \\ \varphi_{2+4j} &= r^{j+2}\sin(j+2)\theta \\ \varphi_{3+4j} &= r^{j+2}\cos(j+2)\theta \\ \varphi_{4+4j} &= r^{j+3}\sin(j+1)\theta \\ \varphi_{5+4j} &= r^{j+3}\cos(j+1)\theta \end{aligned}\right\} \qquad j \geq 0 \ .$$

In this form the series solution is analogous to the the well known series solution for Laplace's equation in a region in the plane, where we include only those terms which are continuous at the origin.

The method thus involves finding an approximation to $\hat{u}$ in the form of a linear combination of the functions, $\{\psi_1,\ldots,\psi_N, \varphi_1,\ldots,\varphi_n, p_1,p_2,p_3\}$, where,

$$\psi_i(t) = (16\pi)^{-1}\|t-t_i\|^2\log\|t-t_i\|^2 \qquad 1 \le i \le N$$

and $\varphi_1,\ldots,\varphi_n$ are the first n functions from the above sequence. Here n is chosen in some way as a *closeness of approximation* parameter and $n = 0$ corresponds simply to the thin plate spline with $\Omega \equiv \mathbf{R}^2$. We return to the choice of n later.

This approximation leaves us with $N+n+3$ parameters to be determined, and the equations.

$$A_\Omega(\hat{u},f) = \sum_{i=1}^{N}\lambda_i f(t_i) \qquad \forall f \in H$$

$$\hat{u}(t_i) = z_i \qquad 1 \le i \le N$$

to use to determine them. The first of these equations we can only hope to satisfy approximately because we are only approximating $\hat{u}$. A technique often used in this situation, is to apply the variational characterisation to the subspace of basis functions, namely, $\{\varphi_1,\ldots,\varphi_n,p_1,p_2,p_3\}$. So letting,

$$\hat{u}_n = \sum_{i=1}^{N}\lambda_i\psi_i + \sum_{i=1}^{n}b_i\varphi_i + \sum_{i=1}^{3}d_ip_i$$

this gives us the following $n+3$ linear equations,

$$\sum_{i=1}^{N}\lambda_i p_j(t_i) = 0 \qquad j = 1,2,3$$

$$\sum_{i=1}^{N}\lambda_i A_\Omega(\psi_i,\varphi_j) + \sum_{i=1}^{n}b_i A_\Omega(\varphi_i,\varphi_j) = \sum_{i=1}^{N}\lambda_i\varphi_j(t_i) \qquad 1 \le j \le n$$

together with the N interpolation conditions,

$$\sum_{i=1}^{N}\lambda_i\psi_i(t_j) + \sum_{i=1}^{n}b_i\varphi_i(t_j) + \sum_{i=1}^{3}d_ip_i(t_j) = z_j \qquad 1 \le j \le N$$

so that we have an $N+n+3$ square linear system to solve to find the unknowns $\lambda_1,\ldots,\lambda_N, b_1,\ldots,b_n, d_1,d_2,d_3$.

In matrix notation with λ,d,T,K,z defined as before and defining,

$$\underset{n\times N}{(B)}_{ij} = \varphi_i(t_j)\ , \qquad \underset{N\times n}{(M_1)}_{ij} = A_\Omega(\psi_i,\varphi_j)\ , \qquad \underset{n\times n}{(M_2)}_{ij} = A_\Omega(\varphi_i,\varphi_j)$$

and

$$\underset{n\times 1}{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

we have the above equations in the partitioned form,

$$T^{\mathrm{T}}\lambda = \mathbf{0} \quad , \quad M_1^{\mathrm{T}}\lambda + M_2 b = B\lambda \quad , \quad K\lambda + B^{\mathrm{T}} b + Td = z,$$

or in the extended matrix form,

$$\begin{bmatrix} K & B^{\mathrm{T}} & T \\ B-M_1^{\mathrm{T}} & -M_2 & 0 \\ T^{\mathrm{T}} & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ b \\ d \end{bmatrix} = \begin{bmatrix} z \\ 0 \\ 0 \end{bmatrix}.$$

The solution is obtained by inverting the positive definite matrix M_2 to write,

$$b = (M_2)^{-1}(B-M_1^{\mathrm{T}})\lambda$$

and thus reducing the system to the first of the above equations together with the following ,

$$(K + B^{\mathrm{T}}(M_2)^{-1}(B-M_1^{\mathrm{T}}))\lambda + Td = z.$$

These equations can then be solved in exactly the same way as before, only remembering that the matrix multiplier of λ is no longer symmetric.

1.3. Smoothing Splines.

We now present a derivation of the form of smoothing splines which follows closely that of Kimeldorf and Wahba (1971).

The bivariate smoothing spline problem is a natural extension of the interpolation problem. Earlier when we considered the interpolation problem it was defined as

Find the $f\in H$ *with* $f(t_i) = z_i \quad (1\le i\le N)$ *such that* $A_\Omega(f,f)$ *is a minimum.*

In the smoothing case we do not require that $f(t_i) = z_i$ only that $f(t_i)$ be in some sense *close* to z_i. How we measure the *closeness* of the function values $f(t_i)$ to the data z_i depends upon the reason for smoothing, however the sum of squared differences is regularly used and is the easiest computationally. So we define the smoothing spline problem by,

$$\textit{Find } f\in H \textit{ with } \sum_{i=1}^{N}(f(t_i)-z_i)^2 + \alpha A_\Omega(f,f) \textit{ a minimum.}$$

Here the scalar parameter $\alpha(>0)$ controls the trade-off between the roughness of the solution and its fidelity to the data. As $\alpha\to 0$ the smoothing spline tends to the interpolating spline and as $\alpha\to\infty$ the smoothing spline tends to the least squares linear regression planar solution.

As before we suppose without loss of generality that t_{N-2}, t_{N-1}, t_N are affine independent, and that p_1, p_2, p_3 are defined as before. Then we define $N-3$ functions by

$$G_i(t) = E_\Omega(t,t_i) - \sum_{j=1}^{3} p_j(t_i)\, E_\Omega(t,t_{N-3+j}) \qquad (1 \le i \le N-3)$$

Notice that

$$G_i(t) = \sum_{j=1}^{N} \omega_{ij}\, E_\Omega(t,t_j)$$

where

$$\left.\begin{aligned} \sum_{j=1}^{N} \omega_{ij}\, p_k(t_j) &= p_k(t_i) - \sum_{l=1}^{3} p_l(t_i)\, p_k(t_{N-3+l}) \\ &= p_k(t_i) - \sum_{l=1}^{3} p_l(t_i)\, \delta_{kl} \\ &= 0 \end{aligned}\right\} \quad (1 \le k \le 3,\ 1 \le i \le N-3)$$

that is

$$\sum_{j=1}^{N} \omega_{ij}\, p(t_j) = 0 \qquad \forall\, p \in \Phi.$$

Therefore each G_i is a member of H and in fact it is easy to see that they are linearly independent since the $E_\Omega(\cdot,t_i)$ are.

From our discussion on interpolatory splines we can see that the functions G_i have a similar property to the interpolant $\hat{u}$, namely that,

$$\begin{aligned} A_\Omega(G_i,f) &= \sum_{j=1}^{N} \omega_{ij} f(t_j) \\ &= f(t_i) - \sum_{j=1}^{3} p_j(t_i) f(t_{N-3+j}) \quad (1 \le i \le N-3) \end{aligned}$$

and reintroducing the projection P onto Φ we have,

$$\begin{aligned} A_\Omega(G_i,f) &= (I-P)(f)(t_i) \\ &= f(t_i) - p_f(t_i) \quad \text{writing } p_f \text{ for } P(f). \end{aligned}$$

Therefore

$$f(t_i) = A_\Omega(G_i,f) + p_f(t_i) \quad (1 \le i \le N)$$

where we define $G_i \equiv 0$ for $i = N-2, N-1, N$.

We can now therefore write the sum of squared differences term from the minimisation functional in the following form,

$$\sum_{i=1}^{N}(f(t_i)-z_i)^2 = \sum_{i=1}^{N}(A_\Omega(G_i,f)+p_f(t_i)-z_i)^2.$$

And then remarking that any function $f\in H$ can be written uniquely as

$$\sum_{i=1}^{N}\eta_i\tilde{G}_i + p_f + g_f$$

where $\tilde{G}_i\equiv(I-P)G_i$. g_f is a function in H with $P(g_f)\equiv 0$ and such that $A_\Omega(G_i,g_f) = 0$ for $1\le i\le N$, we can write the above part of the functional to be minimised in the form,

$$\sum_{i=1}^{N}\{A_\Omega(G_i,\sum_{j=1}^{N}\eta_j\tilde{G}_j+p_f+g_f) + P(\sum_{j=1}^{N}\eta_j\tilde{G}_j+p_f+g_f)(t_i) - z_i\}^2$$
$$= \sum_{i=1}^{N}\{\sum_{j=1}^{N}\eta_j A_\Omega(G_i,G_j) + p_f(t_i) - z_i\}^2$$

since $A_\Omega(G_i,\tilde{G}_j) = A_\Omega(G_i,G_j)$, etc. and since $P(\tilde{G}_j)=0$ etc.

The roughness term in the functional can likewise be written,

$$A_\Omega(\sum_{j=1}^{N}\eta_j\tilde{G}_j+p_f+g_f\,,\,\sum_{j=1}^{N}\eta_j\tilde{G}_j+p_f+g_f)$$
$$= \sum_{i=1}^{N}\sum_{j=1}^{N}\eta_i\eta_j A_\Omega(G_i,G_j) + A_\Omega(g_f,g_f).$$

Combining the above two results into the complete functional to be minimised, we can see that the smallest value will occur when $A_\Omega(g_f,g_f)=0$. Since $P(g_f)=0$ this means that the functional is minimised when $g_f\equiv 0$. This argument shows that the smoothing spline solutions has the same from as the interpolating spline , i.e. the solution is of the form,

$$\sum_{i=1}^{N}\eta_i\tilde{G}_i + p_f$$

or equivalently,

$$\hat{u}_\alpha^{smooth}(t) = \sum_{i=1}^{N}\lambda_i E_\Omega(t,t_i) + \sum_{j=1}^{3}d_j p_j(t)$$

where

$$\sum_{i=1}^{N}\lambda_i p(t_i) = 0 \quad \forall p\in\Phi.$$

So now it only remains to determine the coefficients λ_i and d_j.

We can now rewrite the minimisation functional in terms of the coefficients above. Let $(\tilde{K})_{ij} = E_\Omega(t_i,t_j)$ and T be defined as in the interpolating splines section. Then,

$$\sum_{i=1}^{N} (\hat{u}_\alpha(t_i)-z_i)^2 = (\tilde{K}\lambda+Td-z)^{\mathrm{T}}(\tilde{K}\lambda+Td-z).$$

Also,

$$\begin{aligned} A_\Omega(\hat{u}_\alpha,\hat{u}_\alpha) &= \sum_{i=1}^{N}\lambda_i\hat{u}_\alpha(t_i) \\ &= \sum_{i,j=1}^{N}\lambda_i\lambda_j E_\Omega(t_i,t_j) \\ &= \lambda^{\mathrm{T}}\tilde{K}\lambda \end{aligned}$$

Thus the minimisation functional (for fixed α) is a quadratic form in the coefficients λ_i and d_j, namely,

$$Q(\lambda,d) = (\tilde{K}\lambda+Td-z)^{\mathrm{T}}(\tilde{K}\lambda+Td-z) + \alpha\lambda^{\mathrm{T}}\tilde{K}\lambda.$$

In the case of $\Omega\equiv\mathbf{R}^2$ the matrix $\tilde{K}$ is the same as the matrix K defined in the section on interpolatory splines. However when $\Omega\not\equiv\mathbf{R}^2$ since we cannot find E_Ω exactly the matrix $\tilde{K}$ is unknown. We can approximate $\tilde{K}$ by the matrix corresponding to K in the interpolation equations namely, $K+B^{\mathrm{T}}(M_2)^{-1}(B-M_1^{\mathrm{T}})$ where the matrices B,M_1,M_2 are defined in the earlier section. We now proceed with $\tilde{K}$ as defined above.

It is required to find the extremals of the quadratic form mentioned above. Taking Gâteaux derivatives of the quadratic form gives us the following sets of equations,

$$\begin{aligned} \frac{\partial Q}{\partial\lambda} &= 2\tilde{K}^{\mathrm{T}}\tilde{K}\lambda + 2\tilde{K}^{\mathrm{T}}T - 2\tilde{K}^{\mathrm{T}}z + \alpha(\tilde{K}^{\mathrm{T}}+\tilde{K})\lambda &= 0 \\ \frac{\partial Q}{\partial d} &= 2T^{\mathrm{T}}\tilde{K}\lambda + 2T^{\mathrm{T}}Td - 2T^{\mathrm{T}}z &= 0. \end{aligned}$$

Thus the smoothing spline is completely determined by the form given above and the equations given below,

$$\begin{aligned} (\tilde{K}^{\mathrm{T}}\tilde{K}+\tfrac{1}{2}\alpha(\tilde{K}^{\mathrm{T}}+\tilde{K}))\lambda &= \tilde{K}^{\mathrm{T}}(z-Td) \\ T^{\mathrm{T}}Td &= T^{\mathrm{T}}(z-\tilde{K}\lambda) \\ T^{\mathrm{T}}\lambda &= 0 \end{aligned}$$

In the case $\Omega\equiv\mathbf{R}^2$ Wahba (1979) proposes the following solution of the above equations. Notice that in this case the matrix $\tilde{K}$ is symmetric and if we define the matrix U as in the section on interpolation, and demand that $U^{\mathrm{T}}U=I$, then defining $\bar{K} = U^{\mathrm{T}}\tilde{K}U$ the above equations reduce to,

$$\begin{aligned}(\bar{K}+\alpha I)\gamma &= U^t z \\ \lambda &= U\gamma \\ T^{\mathrm{T}}Td &= T^{\mathrm{T}}(z-\tilde{K}\lambda).\end{aligned}$$

Now $\bar{K}$ is real symmetric matrix and therefore diagonalisable, so writing $\bar{K} = RDR^{\mathrm{T}}$ we have (for all $\alpha \geq 0$),

$$\lambda = UR(D+\alpha I)^{-1}R^{\mathrm{T}}U^{\mathrm{T}}z$$

Using this form of the equations, to find a smoothing spline solution for sequence of different α's it is only necessary to invert a diagonal matrix and carry out some matrix multiplication for each new value of α.

We are motivated by this result to find an equivalent form for the equations when $\Omega \neq \mathbf{R}^2$, that is when the matrix $\tilde{K}$ is not necessarily (or only approximately) symmetric. Solving the second set of equations for d we find that $Td = T(T^{\mathrm{T}}T)^{-1}T^{\mathrm{T}}(z-\tilde{K}\lambda)$. Notice that $T(T^{\mathrm{T}}T)^{-1}T^{\mathrm{T}}$ is the projection matrix onto the subspace of $\mathbf{R}^N$ spanned by the columns of T. Notice also that since $U^{\mathrm{T}}U = I_{N-3}$ the matrix UU^{T} represents a projection and since $U^{\mathrm{T}}T = 0$ and the rank of U is $N-3$, this projection is onto the orthogonal complement of the subspace of $\mathbf{R}^N$ spanned by the columns of T. This means that,

$$T(T^{\mathrm{T}}T)^{-1}T^{\mathrm{T}} = I - UU^{\mathrm{T}}.$$

Using this result and substituting into the first set of equations we have,

$$(\tilde{K}^{\mathrm{T}}\tilde{K}+\tfrac{1}{2}\alpha(\tilde{K}^{\mathrm{T}}+\tilde{K}))\lambda = \tilde{K}^{\mathrm{T}}(z-(I-UU^{\mathrm{T}})(z-\tilde{K}\lambda))$$

therefore

$$((\tilde{K}^{\mathrm{T}}U)(U^{\mathrm{T}}\tilde{K})+\tfrac{1}{2}\alpha(\tilde{K}^{\mathrm{T}}+\tilde{K}))\lambda = (\tilde{K}^{\mathrm{T}}U)(U^{\mathrm{T}}z).$$

Letting $\lambda = U\gamma$ and $\bar{K}$ defined as above the equations become,

$$(\bar{K}^{\mathrm{T}}\bar{K}+\tfrac{1}{2}\alpha(\bar{K}^{\mathrm{T}}+\bar{K}))\gamma = \bar{K}^{\mathrm{T}}(U^{\mathrm{T}}z).$$

These equations have a somewhat similar form to those derived when $\Omega \equiv \mathbf{R}^2$, but are more complicated in the sense that $\bar{K}^{\mathrm{T}}+\bar{K}$ a real symmetric matrix replaces I as the multiplier of α. This means of course that the solution of these equations is slightly more complicated.

Firstly we note that since $\bar{K}^{\mathrm{T}}\bar{K}$ is a real symmetric positive definite matrix it can be written in the form SS^{T} for several *nice* choices of S. For example S might be a lower triangular matrix or of the form R_1D_1 where R_1 is an orthogonal matrix and D_1 is a diagonal matrix. We must choose a decomposition that is computationally easiest to work with. Then the above equations can be written,

$$S(I+\tfrac{1}{2}\alpha S^{-1}(\bar{K}^{\mathrm{T}}+\bar{K})S^{-\mathrm{T}})S^{\mathrm{T}}\gamma = \bar{K}^{\mathrm{T}}(U^{\mathrm{T}}z).$$

Diagonalising $\frac{1}{2}S^{-1}(\bar{K}^{\mathrm{T}}+\bar{K})S^{-\mathrm{T}}$ we can now proceed as before.

In this way more work is needed initially but then in order to calculate γ for a sequence of values of α very little extra work is required over that for $\Omega \equiv \mathbf{R}^2$.

Chapter 2.
Details of the Implementation of Bivariate Splines.

In this chapter we will describe some of the details of our implementation of bivariate splines, where it differs from the implementations of Wahba and of Dyn and Levin.

2.1. Splines with $\Omega \equiv \mathbf{R}^2$.

There is only one point of major interest in the implementation of interpolatory splines when $\Omega \equiv \mathbf{R}^2$, namely the calculation of the matrix U. Wahba (1979) uses the eigenvectors of the matrix $T(T^{\mathrm{T}}T)^{-1}T^{\mathrm{T}}$ corresponding to the eigenvalues which are 0 ($N-3$ of them), as the columns of U. This involves the diagonalisation of an $N \times N$ matrix, however, which Wahba treats as an ordinary symmetric matrix, and we try to find these eigenvectors using the special form of T.

Our method consists of completing the set of vectors formed by the columns of T into a basis for $\mathbf{R}^N$ and using the Gram-Schmidt orthogonalisation process to orthonormalise this basis. The Gram-Schmidt process is generally regarded as numerically unstable because of the large number of inner products involved, and because of the way in which errors accumulate as the process proceeds. Since in our case most of the inner products are trivial these errors are significantly reduced. In fact the only full inner products required are at the normalisation stage of each vector and these can be carried out in extra precision. In the case of interpolatory splines it is not necessary that that $U^{\mathrm{T}}U=I$ only that $U^{\mathrm{T}}T=0$, and the columns of U are linearly independent. In the smoothing case, where we also need U, the orthonormality condition is necessary in the current formulation, (it could be eliminated at the cost of some other computational work), so here we orthonormalise the vectors. Thus we can see that some of the errors are unimportant in at least one of our situations.

The first step is to extend the columns of T to a basis for $\mathbf{R}^N$. The matrix T has three columns given by the three vectors,

$$T_1 = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}, \; T_x = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}, \; T_y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}.$$

(Where here $(x_i, y_i) = t_i$ are the sites at which data are observed.)

First we notice that non-collinearity of the data sites implies that at least one triple of points is not collinear. Without loss of generality we may assume that this triple is, (x_1,y_1), (x_2,y_2) and (x_3,y_3). It may be desirable for numerical reasons to reorder the points such that this triple is actually markedly non-collinear.

With this possible reordering assumed, now define a basis for $\mathbf{R}^N$, $\{w_i\}$, by setting $w_1 = T_1$, $w_2 = T_x$, $w_3 = T_y$ and for $i=4,\ldots,N$ let w_i be that vector with a 1 in the ith position and zeros elsewhere. $w_1,\ldots,w_N$ then form a basis. Suppose $\sum_i \mu_i w_i = 0$ then because the 3×3 matrix in the top left hand corner has been made nonsingular, and the rest of the top three rows are zero, we have $\mu_1=\mu_2=\mu_3=0$, and it follows immediately that $\mu_4=\ldots=\mu_N=0$ from the form of $w_4,\ldots,w_N$.

We can now orthonormalise this basis using the Gram-Schmidt process. For each k the vector u_k is obtained from w_k by setting

$$\begin{aligned} v_k &= w_k - \sum_{i=1}^{k-1} (w_k^{\mathrm{T}} u_i)\, u_i \\ u_k &= v_k / \sqrt{(v_k^{\mathrm{T}} v_k)}\,. \end{aligned}$$

Notice that since each w_k has only one non-zero entry when $k\geq 4$, the inner products of each w_k with one of the previously orthonormalised vectors, reduces to picking out a component of the vector. Thus the only full inner product required is that of a vector with itself, in order to normalise it, and this can be carried out in extra precision if desired. This also means that this version of the process is $O(N^2)$ instead of the usual $O(N^3)$ for a full matrix diagonalisation.

Since after orthogonalising the vectors u_1,u_2,u_3 span the same space as w_1,w_2,w_3 which are just T_1,T_x,T_y respectively, we can take the last $N-3$ vectors of this new basis as the columns of U.

Notice also that what we have effectively done to the matrix T is find a 3×3 invertible matrix A such that the columns of TA are orthonormal. In fact A is upper triangular. Multiplying this new matrix by its transpose we therefore find that,

$$I = (TA)^{\mathrm{T}} TA$$

or equivalently

$$AA^{\mathrm{T}} = (T^{\mathrm{T}}T)^{-1}.$$

This means that if we save the details of the transformation A we can compute the inverse of $T^{\mathrm{T}}T$ for the extra work of one 3×3 triangular matrix multiplication. (Note that this is without even explicitly computing $T^{\mathrm{T}}T$.)

Professor R.Sibson has suggested an alternative means of finding a U suitable for the interpolation case immediately, without orthonormalising, and suitable for the smoothing case after orthonormalisation. Given 3 points which are not collinear, without loss of generality t_1, t_2 and t_3, we can generate $N-3$ vectors which are linearly independent and orthogonal to the columns of T, in the following way. For each $j > 3$ find the homogeneous coordinates of t_j with respect to the triangle formed by t_1, t_2 and t_3. Call these coordinates w_{ij}, where $i = 1,2,3$, so that we have,

$$t_j = \sum_{i=1}^{3} w_{ij} t_i .$$

Since these coordinates are homogeneous we have that $w_{1j}+w_{2j}+w_{3j} = 1$ for each j so defining the N-vector u_j by,

$$(u_j)_i = \begin{cases} w_{ij} & i = 1,2,3 \\ -1 & i = j \\ 0 & \text{otherwise,} \end{cases}$$

we see that,

$$\begin{aligned} T^{\mathrm{T}} u_j &= \left(\sum_{i=1}^{N} (u_j)_i, \sum_{i=1}^{N} (u_j)_i x_i, \sum_{i=1}^{N} (u_j)_i y_i\right)^{\mathrm{T}} \\ &= (w_{1j}+w_{2j}+w_{3j}-1, w_{1j}x_1+w_{2j}x_2+w_{3j}x_3-x_j, w_{1j}y_1+w_{2j}y_2+w_{3j}y_3-y_j)^{\mathrm{T}} \\ &= \mathbf{0} \end{aligned}$$

since w_{ij} are homogeneous coordinates. Thus the matrix whose columns are the u_j's will do as a U in the interpolation case. Notice that this U is sparse, as the first three rows are full, while the last $N-3$ rows are the negative of the $N-3$ by $N-3$ identity matrix, which is not the form that would arise if the previous method were stopped before the columns of U were orthonormalised.

2.2 The Condition of the Linear System.

The practical application of thin plate splines is known to be restricted, using the techniques discussed earlier, to approximately 200 points. This is due to the numerical ill-conditioning and denseness of the linear system arising out of the problem. (See Wahba (1979), Dyn and Levin (1983) for example). In order to fit splines to larger numbers of data sites it may be necessary to use some iterative means of solving this linear system. Many of these methods are sensitive to the conditioning of the system, and appropriate preconditioning may significantly affect the ease of solution. (For an example see the conjugate gradient algorithm in later chapters). In this section we shall discuss the conditioning of the linear system involved in the thin plate spline problem, and mention one possible way of

improving this condition.

Recall that the linear system to be solved can be written in the form,

$$\begin{bmatrix} K & T \\ T^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ d \end{bmatrix} = \begin{bmatrix} z \\ 0 \end{bmatrix}.$$

It is the condition of this system that we shall be concerned with here. This means of course that we are essentially looking at the interpolation problem, but the smoothing case is similar with the equivalent system,

$$\begin{bmatrix} \alpha I+K & T \\ T^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ d \end{bmatrix} = \begin{bmatrix} z \\ 0 \end{bmatrix}.$$

How the addition of this *ridge* term affects the conditioning of the problem we shall not consider here.

In the examples which follow we will measure the condition of a matrix, A (say), by the ratio $\sqrt{\mu_{\max}}/\sqrt{\mu_{\min}}$ where $\mu_{\max}$ and $\mu_{\min}$ are the largest and smallest eigenvalues of the matrix $A^{\mathrm{T}}A$. In the case that A is symmetric this is just the absolute value of the ratio of the largest and smallest (in absolute value) eigenvalues of A, and corresponds to the well known condition number.

As example matrices we have used those from the linear systems associated with six interpolation problems. These are 49, 100 and 225 points scattered randomly in the square, $0 \le x, y \le 1$ and the corresponding 7×7, 10×10 and 15×15 grids of points in the same square.

Before we discuss the condition numbers of the above systems, let us investigate some possible ways of improving their conditioning. Remember that the matrix K above arises from the basis functions E_i, in that $K_{ij} = E_j(t_i)$. So the ith row of the augmented matrix corresponds to the values of all the basis functions (including the polynomial basis $1, x, y$) at the point t_i. For each j the basis function $E_j(t)$ is proportional to $r_j^2 \log r_j^2$ where r_j^2 is the squared distance of t from t_j. These functions all increase at a rate greater than x^2+y^2 as the point $t=(x,y)$ approaches infinity, so the off diagonal entries in K may be very large indeed. Notice also that the diagonal entries in K are zero, so that this effect will probably contribute to the ill-conditioning of the augmented matrix in the linear system. We might, therefore, try and improve the condition by improving the diagonal dominance in K and this is achieved via a change of basis, that reduces the rate of growth of the basis functions.

A change of basis can be expressed as,

$$F_i(t) = \sum_j w_{ij} E_j(t) \quad i=1,\ldots,N$$

where $W=(w_{ij})$ is a non-singular matrix. It now remains to find a suitable choice for the *weights* w_{ij}. Before proposing such a system of weights first let us look at the behaviour of the E_j. Recall that,

$$16\pi E_j(t) = \|t-t_j\|^2 \log\|t-t_j\|^2$$

where $t=(x,y)$ and $t_j=(x_j,y_j)$. Now we see that,

$$\begin{aligned}\log\|t-t_j\|^2 &= \log(\|t\|^2 - 2(t^{\mathrm{T}}t_j) + \|t_j\|^2) \\ &= \log\|t\|^2 + \log(1+\frac{\|t_j\|^2 - 2t^{\mathrm{T}}t_j}{\|t\|^2})\end{aligned}$$

so that

$$\log\|t-t_j\|^2 = \log\|t\|^2 + \sum_{i=1}^{\infty} \frac{(-1)^{i-1}}{i} (\frac{\|t_j\|^2 - 2t^{\mathrm{T}}t_j}{\|t\|^2})^i$$

provided $|\|t_j\|^2 - 2t^{\mathrm{T}}t_j| < \|t\|^2$. (In fact we require $|\|t_j\|^2 - 2t^{\mathrm{T}}t_j|$ much less than $\|t\|^2$ for the series to converge usefully fast.)

Thus

$$\begin{aligned}16\pi E_j(t) = \|t\|^2 [\log\|t\|^2 &+ \frac{\|t_j\|^2 - 2t^{\mathrm{T}}t_j}{\|t\|^2} - \frac{1}{2}(\frac{\|t_j\|^2 - 2t^{\mathrm{T}}t_j}{\|t\|^2})^2] \\ &- (2t^{\mathrm{T}}t_j)[\log\|t\|^2 + \frac{\|t_j\|^2 - 2t^{\mathrm{T}}t_j}{\|t\|^2}] \\ &+ \|t_j\|^2 \log\|t\|^2 + O(\frac{1}{\|t\|}) \quad \text{as } \|t\| \to \infty\end{aligned}$$

So that, given a set of weights $\{w_{ij}\}$, we have,

$$\begin{aligned}16\pi \sum_j w_{ij} E_j(t) = &(\sum_j w_{ij}) \|t\|^2 \log\|t\|^2 - 2(1+\log\|t\|^2) t^{\mathrm{T}} (\sum_j w_{ij} t_j) \\ &+ (\sum_j w_{ij} \|t_j\|^2)(1+\log\|t\|^2) + 2\sum_j \frac{w_{ij}(t^{\mathrm{T}}t_j)^2}{\|t\|^2} + O(\frac{1}{\|t\|})\end{aligned}$$

Suppose now that the weights are such that,

$$\sum_j w_{ij} = \sum_j w_{ij} t_j = 0 \quad \text{for nearly all } i. \qquad (*)$$

(If the above were to hold for all i then (w_{ij}) would be singular and therefore not a change of basis.)

In this case we have, for those i for which (*) holds,

$$16\pi\sum_j w_{ij}E_j(t) = (\sum_j w_{ij}\|t_j\|^2)(1+\log\|t\|^2)+2\frac{\sum_j w_{ij}(t^{\mathrm{T}}t_j)^2}{\|t\|^2}+O(\frac{1}{\|t\|})$$

since (*) makes the first two terms vanish. This change of basis has therefore reduced the rate of growth to that of $\log\|t\|$. Translating the origin and noticing that, $\sum_j w_{ij}(t_j-t_i) = 0$ for those i for which the above mentioned conditions on w_{ij} are satisfied, we see that,

$$16\pi\sum_j w_{ij}E_j(t) = (\sum_j w_{ij}\|t_j-t_i\|^2)(1+\log\|t-t_i\|^2)$$

$$+2\|t-t_i\|^{-2}\sum_j w_{ij}((t-t_i)^{\mathrm{T}}(t_j-t_i))^2 + O(\frac{1}{\|t\|})$$

for such i.

With this new basis, since $\sum\lambda'_i F_i$ is just $\sum_j(\sum_i w_{ij}\lambda'_i)E_j$, the linear system to be solved becomes,

$$\begin{aligned} KW^{\mathrm{T}}\lambda' + Td &= z \\ T^{\mathrm{T}}W^{\mathrm{T}}\lambda' &= 0 \end{aligned}$$

so that the condition of the new problem is thus the condition of the matrix,

$$\begin{bmatrix} KW^{\mathrm{T}} & T \\ T^{\mathrm{T}}W^{\mathrm{T}} & 0 \end{bmatrix}.$$

It only remains therefore to define a set of weights, that satisfy the above. Such a set of weights can be derived from the boundary over distance weights or subtile weights arising from the Dirichlet tessellation of the data sites, and described in Chapter 4 and in Sibson (1985) and Sibson (1980). As shown in Sibson (1980, 1985) both these sets of weights satisfy the conditions,

$$\sum_{j\neq i} v_{ij}(t_j-t_i) = 0$$

for all i such that t_i is an interior point of the data. So we can define w_{ij}, the change of basis matrix by,

$$w_{ij} = \begin{cases} v_{ij} & j\neq i \\ -\sum_{k\neq i} v_{ik} & j=i \end{cases}$$

and these satisfy all the properties mentioned above. These weights as derived are sparse, in the sense that w_{ij} is non-zero only when t_i and t_j are neighbours (as

defined by the Dirichlet tessellation). Also the above statements about the resulting rate of growth of the new basis F_i, hold whenever t_i has a complete set of neighbours (ie. is not a neighbour of the boundary).

The above derived augmented matrix for the new basis is not in general symmetric, so it might be a good idea to make this matrix symmetric by premultiplying the first set of equations by W. Thus we get the linear system,

$$\begin{aligned} WKW^{\mathrm{T}}\lambda' + WTd &= Wz \\ T^{\mathrm{T}}W^{\mathrm{T}}\lambda' &= 0 \end{aligned}$$

and the associated condition is that of the matrix,

$$\begin{bmatrix} WKW^{\mathrm{T}} & WT \\ (WT)^{\mathrm{T}} & 0 \end{bmatrix},$$

a symmetric matrix. This matrix can be written as a product of three matrices in the following form,

$$\begin{bmatrix} W & 0 \\ 0 & I_3 \end{bmatrix} \begin{bmatrix} K & T \\ T^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} W^{\mathrm{T}} & 0 \\ 0 & I_3 \end{bmatrix}$$

which is similar to (and therefore has the same eigenvalues as),

$$\begin{bmatrix} K & T \\ T^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} W^{\mathrm{T}} & 0 \\ 0 & I_3 \end{bmatrix} \begin{bmatrix} W & 0 \\ 0 & I_3 \end{bmatrix}.$$

This matrix can be written,

$$\begin{bmatrix} KW^{\mathrm{T}}W & T \\ T^{\mathrm{T}}W^{\mathrm{T}}W & 0 \end{bmatrix}.$$

Thus we see that the condition of the symmetric linear system above is just the condition of the above unsymmetric matrix. This unsymmetric matrix is just that which is obtained by a second change of basis, using the transpose of the weighting matrix used in the first change of basis. So in the case that the weighting matrix is symmetric, as it is in the case of the weights derived from the boundary over distance weights, this is just a repeat of the first change of basis. The weights derived from the subtile weights are not symmetric and so the above does not apply.

Before we can get the best advantage from this second change of basis, however, it is necessary to ensure that all the F_i have similar leading terms. Recall that each F_i has a fastest growing term of the form,

$$(\sum_j w_{ij} \| t_j - t_i \|^2) \log \| t - t_i \|^2$$

so that we can rescale these basis functions to have similar leading terms. Put

$D_{ii} = (\sum_k w_{ik} \| t_k - t_i \|^2)^{-1}$ for each i and $D_{ij} = 0$ whenever $i \neq j$. Then this rescaling can be accomplised using this matrix D for a change of basis, between the two changes of basis using the matrix W.

Putting $G_i(t) = \sum_k w_{ik} D_{kk} F_k(t)$, and using the same arguments as before in is easy to see that,

$$G_i(t) = (\sum_k w_{ik}) \log \| t - t_i \|^2 + O(1) = O(1) \qquad \text{as } \| t \| \to \infty,$$

provided that t_i is not a neighbour of the boundary, and has no neighbours which neighbour the boundary. Thus the rapid growth in E_i has been completely removed.

In terms of the original linear system, the matrix which is associated with this condition is given by,

$$\begin{bmatrix} D^{\frac{1}{2}} W K W^{\mathrm{T}} D^{\frac{1}{2}} & D^{\frac{1}{2}} W T \\ (D^{\frac{1}{2}} W T)^{\mathrm{T}} & 0 \end{bmatrix}.$$

In order to understand this behaviour more clearly, and get some impression of the behaviour of all the different basis functions closer to the points t_i, (since all the above results show the behaviour of the basis functions as $\| t \| \to \infty$), we have calculated the basis functions for a particular example. We have used four points at the corners of a square and a central point at the intersection of the diagonals of the square. We have then drawn a cross section through the three basis functions, E_1, F_1 and G_1 where t_1 is taken as the central point. (Extending the grid as necessary to define G_1). Figure 2.2.1 shows the cross-sections through the basis functions where the solid line is E_1, the dotted line is F_1 and the dot-dashed line is G_1. All the cross-sections have been rescaled so that their range is 1.0 over the plotted area, so that we may compare rates of growth more easily. Notice that E_1 grows rapidly as the distance away from the central point increases, whereas F_1 increases more slowly, and G_1 actually decays away to zero. This suggests that we should be able to say more about G_1 than $G_1(t) = O(1)$ as t approaches infinity. However the grided example is rather a special case. Figure 2.2.2 shows a similar picture to the above but here the points upon which the basis functions a based are scattered more arbitrarily. Here we again see the faster than quadratic growth of E_1, the logarithmic growth of F_1, but now G_1 does not decay to zero; it appears to tend to a constant. This suggests that the gridded example, where G_1 does tend to zero, is indeed a special case.

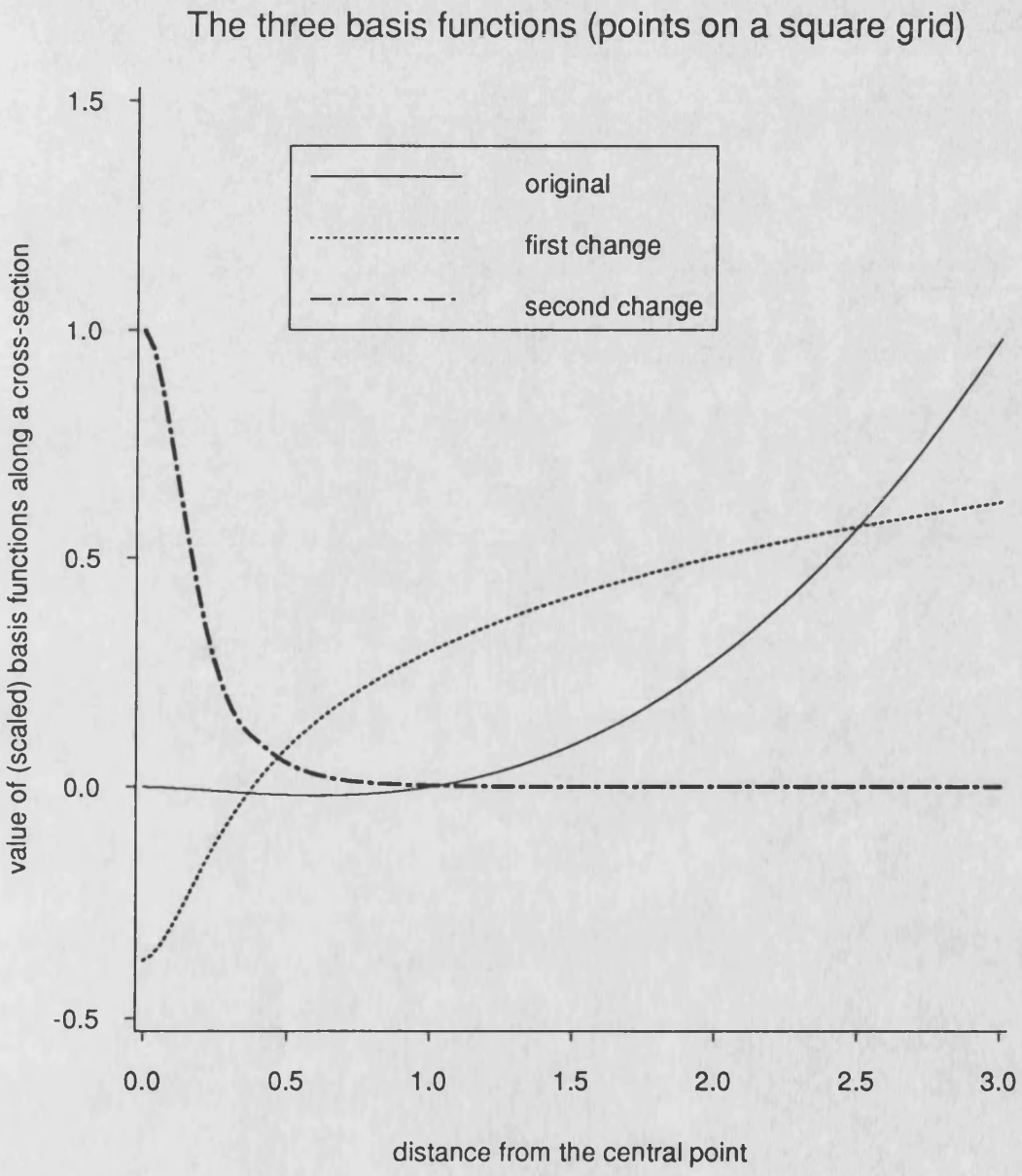

The three basis functions (points on a square grid)
original
first change
second change
value of (scaled) basis functions along a cross-section
1.5
1.0
0.5
0.0
-0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
distance from the central point

figure 2.2.1

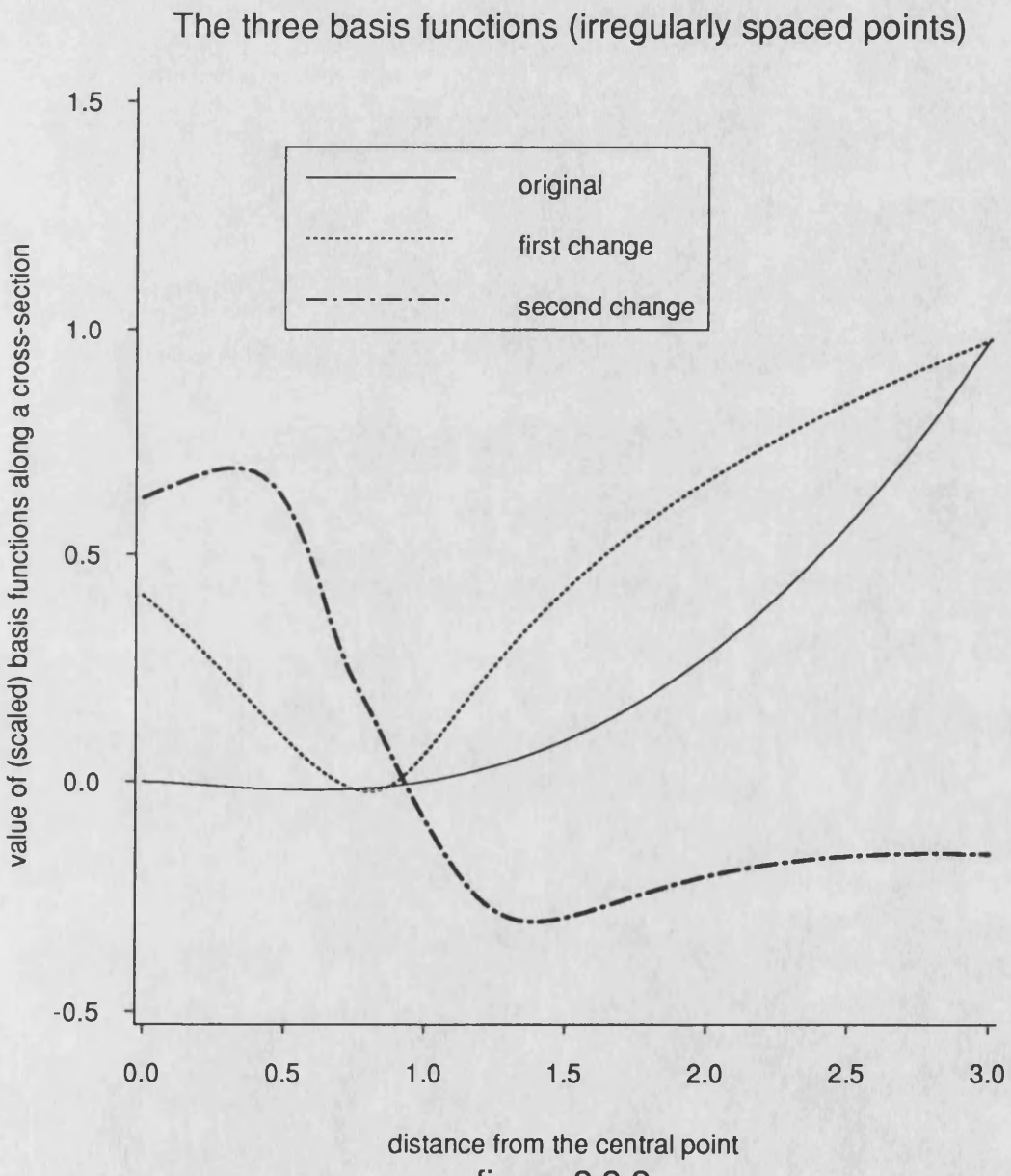

The three basis functions (irregularly spaced points)
original
first change
second change
value of (scaled) basis functions along a cross-section
1.5
1.0
0.5
0.0
-0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
distance from the central point

figure 2.2.2

We can prove this property for gridded data by considering the $O(1)$ term in the expansion. D_{ii} is constant for all i (away from the boundary) in this case so we can ignore the rescaling step. The $O(1)$ term in G_i is then proportional to,

$$\sum_j w_{ij} \| t-t_j \|^{-2} \sum_k w_{jk} ((t-t_j)^{\mathrm{T}} (t_k-t_j))^2 .$$

This term can be rewritten in the form,

$$\sum_j w_{ij} \frac{(t-t_j)^{\mathrm{T}} (\sum_k w_{jk} (t_k-t_j)(t_k-t_j)^{\mathrm{T}})(t-t_j)}{(t-t_j)^{\mathrm{T}} (t-t_j)}$$

In the case of gridded data the 2×2 matrix in the numerator of the above is the same for all j provided only that t_j is away from the boundary. For all regular grids (excluding rectangular) symmetry arguments imply that,

$$\sum_k w_{jk} (t_k-t_j)(t_k-t_j)^{\mathrm{T}} \propto \sum_k (t_k-t_j)(t_k-t_j)^{\mathrm{T}}$$

Symmetry also gives us,

$$\sum_k (x_k-x_j)(y_k-y_j) = 0$$

$$\sum_k (x_k-x_j)^2 = \sum_k (y_k-y_j)^2$$

so that the 2 by 2 matrix is a multiple of the identity matrix. Thus we see that the $O(1)$ term in G_i is proportional to, $\sum_j w_{ij}$ which vanishes whenever t_i is not on the boundary.

The case of rectangular grids can be proved as follows. Suppose that the grid has dimensions a in the x direction and b in the y direction. Then the matrices $(t_k-t_j)(t_k-t_j)^{\mathrm{T}}$ have one of the two forms,

$$\begin{bmatrix} a^2 & 0 \\ 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & 0 \\ 0 & b^2 \end{bmatrix}.$$

and the corresponding boundary over distance weights are, $b/2a$ and $a/2b$. Thus the 2 by 2 matrix is,

$$2(b/2a) \begin{bmatrix} a^2 & 0 \\ 0 & 0 \end{bmatrix} + 2(b/2a) \begin{bmatrix} 0 & 0 \\ 0 & b^2 \end{bmatrix} = (ab) I_2 .$$

So again the $O(1)$ term is proportional to, $\sum_j w_{ij} = 0$.

Thus we see that G_i is in fact $O(1/\| t \|)$ when the data is gridded and t_i is not a neighbour of the boundary and has no neighbours neighbouring the boundary.

The table 2.2.1 shows the condition numbers of the various linear systems associated with the problems mentioned earlier, where the weights are those derived from the boundary over distance weights.

Table 2.2.1.

	No. of Points	Preconditioning None	Unsymmetric	Symmetric
Scattered	49	5.5×10^5	2.0×10^5	4.1×10^3
	100	1.0×10^7	1.1×10^6	6.5×10^3
	225	1.6×10^7	2.1×10^6	2.8×10^4
Gridded	49	4.5×10^4	3.5×10^3	1.9×10^2
	100	1.3×10^5	7.3×10^3	3.1×10^2
	225	4.2×10^5	1.7×10^4	5.4×10^2

These figures represent reductions, when using the symmetric conditioning, by factors of 130, 1500 and 570 respectively in the scattered examples, and 230, 420 and 770 respectively in the gridded examples.

2.3. Interpolatory splines with $\Omega \neq R^2$.

We now turn to the calculation of interpolating splines over regions Ω that are not the whole plane. Remember that we have restricted ourselves to, not necessarily convex, polygonal regions for simplicicty, so that a region can be specified in terms of the coordinates in the plane of its vertices.

Given such a region we must calculate the matrices given by,

$$(M_1)_{ij} = A_\Omega(\psi_i,\varphi_j)\ ,\ (M_2)_{ij} = A_\Omega(\varphi_i,\varphi_j).$$

We proceed first with the calculation of M_2. Recall that,

$$A_\Omega(u,v) = \int_\Omega (\nabla^4 u)v + \oint_{\partial\Omega}\{\nabla(\partial_n u).\nabla v - (\partial_n \nabla^2 u)v\}$$

so that in our current case of interest we have, since $\nabla^4\varphi \equiv 0$ for all φ functions,

$$(M_2)_{ij} = A_\Omega(\varphi_i,\varphi_j) = \oint_{\partial\Omega}\{\nabla(\partial_n\varphi_i).\nabla\varphi_j - (\partial_n\nabla^2\varphi_i)\varphi_j\}.$$

The area integral has therefore been converted to a contour integral which since we are interested in polygonal regions, and the integrands are sufficiently well behaved, reduces to a sum of line integrals. Dyn and Levin propose the calculation of these line integrals using Simpson's rule and the numerical approach was that first adopted by us. A much faster, if conceptually more complicated, approach has also been implemented by us and we will discuss this later.

In order to use a numerical rule to evaluate the integrals we must be able to evaluate the integrand at any point on $\partial\Omega$. Thus we require expressions for the derivatives,

$$\nabla\varphi_i \ , \ \nabla(\partial_n\varphi_i) \ , \text{ and } \ \partial_n\nabla^2\varphi_i .$$

Now in our case we are interested in piecewise linear boundaries and the derivative ∂_n mentioned above reduces to $n_x\frac{\partial}{\partial x}+n_y\frac{\partial}{\partial y}$ on each piece of the boundary, (where (n_x,n_y) is the unit normal to that piece of boundary). Thus the inner product of φ_i with φ_j reduces to a sum of integrals of the forms,

$$n_x\int\frac{\partial^2\varphi_i}{\partial x^2}\frac{\partial\varphi_j}{\partial x}+\frac{\partial^2\varphi_i}{\partial x\partial y}\frac{\partial\varphi_j}{\partial y}-(\frac{\partial}{\partial x}\nabla^2\varphi_i)\varphi_j$$

and

$$n_y\int\frac{\partial^2\varphi_i}{\partial y^2}\frac{\partial\varphi_j}{\partial y}+\frac{\partial^2\varphi_i}{\partial x\partial y}\frac{\partial\varphi_j}{\partial x}-(\frac{\partial}{\partial y}\nabla^2\varphi_i)\varphi_j$$

where each integral is along a straight line forming part of the boundary of Ω and is with respect to arc length.

So we actually require all the first and second derivatives and the gradient of the Laplacian of the φ functions on the boundary of Ω. Define first the functions,

$$c_j(r,\theta)=r^j\cos j\theta \ \text{ and } \ s_j(r,\theta)=r^j\sin j\theta \ \ \forall j\geq 0.$$

All the φ functions can be expressed simply in terms of these functions and r^2, in the following way.

$$\left.\begin{aligned}\varphi_1 &= r^2\\ \varphi_{2+4j} &= s_{j+2}\\ \varphi_{3+4j} &= c_{j+2}\\ \varphi_{4+4j} &= r^2s_{j+1}\\ \varphi_{5+4j} &= r^2c_{j+1}\end{aligned}\right\} \ j\geq 0.$$

Notice also that letting $z=x+iy$ we have,

$$\frac{\partial c_j}{\partial x}=\frac{\partial}{\partial x}\mathrm{Re}(z^j)=j\mathrm{Re}(z^{j-1})=jc_{j-1}$$

And using similar means we get the complete set of derivatives,

$$\frac{\partial c_j}{\partial x}=jc_{j-1} \qquad \frac{\partial s_j}{\partial x}=js_{j-1}$$

$$\frac{\partial c_j}{\partial y}=-js_{j-1} \qquad \frac{\partial s_j}{\partial y}=jc_{j-1}$$

Using these results and the above definitions of the φ functions we get the following set of derivatives. (Here H(f) represents the Hessian matrix of the function f).

φ_1

$$\nabla\varphi_1 = 2\begin{bmatrix} c_1 \\ s_1 \end{bmatrix}, \; \mathrm{H}(\varphi_1) = 2I_2 \; , \; \nabla(\nabla^2\varphi_1) = \mathbf{0} \; ,$$

φ_{2+4j}

$$\nabla\varphi_{2+4j} = (j+2)\begin{bmatrix} s_{j+1} \\ c_{j+1} \end{bmatrix}, \; \mathrm{H}(\varphi_{2+4j}) = (j+2)(j+1)\begin{bmatrix} s_j & c_j \\ c_j & -s_j \end{bmatrix},$$

$$\nabla(\nabla^2\varphi_{2+4j}) = \mathbf{0} \; ,$$

φ_{3+4j}

$$\nabla\varphi_{3+4j} = (j+2)\begin{bmatrix} c_{j+1} \\ -s_{j+1} \end{bmatrix}, \; \mathrm{H}(\varphi_{3+4j}) = (j+2)(j+1)\begin{bmatrix} c_j & -s_j \\ -s_j & -c_j \end{bmatrix},$$

$$\nabla(\nabla^2\varphi_{3+4j}) = \mathbf{0} \; ,$$

φ_{4+4j}

$$\nabla\varphi_{4+4j} = \begin{bmatrix} s_{j+2}+(j+2)r^2 s_j \\ -c_{j+2}+(j+2)r^2 c_j \end{bmatrix},$$

$$\mathrm{H}(\varphi_{4+4j}) = (j+2)\begin{bmatrix} 2s_{j+1}+(j+1)r^2 s_{j-1} & (j+1)r^2 c_{j-1} \\ (j+1)r^2 c_{j-1} & 2s_{j+1}-(j+1)r^2 s_{j-1} \end{bmatrix},$$

$$\nabla(\nabla^2\varphi_{4+4j}) = 4(j+2)(j+1)\begin{bmatrix} s_j \\ c_j \end{bmatrix},$$

φ_{5+4j}

$$\nabla\varphi_{5+4j} = \begin{bmatrix} c_{j+2}+(j+2)r^2 c_j \\ s_{j+2}-(j+2)r^2 s_j \end{bmatrix},$$

$$\mathrm{H}(\varphi_{5+4j}) = (j+2)\begin{bmatrix} 2c_{j+1}+(j+1)r^2 c_{j-1} & -(j+1)r^2 s_{j-1} \\ -(j+1)r^2 s_{j-1} & 2c_{j+1}-(j+1)r^2 c_{j-1} \end{bmatrix},$$

$$\nabla(\nabla^2\varphi_{5+4j}) = 4(j+2)(j+1)\begin{bmatrix} c_j \\ -s_j \end{bmatrix},$$

where here we have introduced the extra definitions, $c_{-1} = 0$ and $s_{-1} = 0$ for

notational convenience.

These derivatives are all expressed in terms of the functions c_j, s_j and r^2, and so are easily calculated once we have the above functions. It is important to have efficient ways of calculating the c_j and s_j since it is likely that they will be called often when calculating the line integrals by numerical means. If there is an efficient implementation of the sine, cosine and integral power functions available, the c_j and s_j can be computed using the formulae given as their definitions.

There is however a pair of recurrence relations that can be used for calculating these functions, (in cartesian coordinates), namely,

$$\begin{aligned} c_{j+1}(x,y) &= xc_j(x,y) - ys_j(x,y) \\ s_{j+1}(x,y) &= xs_j(x,y) + yc_j(x,y) \end{aligned}$$

and we prefer to use these, since we often require the values of many φ functions at the same point.

For the integration algorithm itself Dyn and Levin report using Simpson's rule which is adequate for this case of calculating the elements of M_2 since the φ functions are actually only moderately low degree bivariate polynomials. However we found this rule to be a little slow to reach a given accuracy when calculating the elements of M_1, when the integrands involve logarithms, and so we used Romberg integration. Greater efficiency could probably be achieved using a Gaussian integration rule, (probably with logarithmic weights), but we did not investigate this since we have a much more efficient means of evaluating the integrals.

We consider now the evaluation of the elements of M_1, that is $A_\Omega(\psi_i,\varphi_j)$. Using the symmetry of the semi-inner product and a similar subsequent argument to that used before we see that we actually only require to evaluate ψ and $\nabla\psi$ on the boundary of Ω. Now,

$$\psi_i(x,y) = (16\pi)^{-1}((x-x_i)^2+(y-y_i)^2)\log((x-x_i)^2+(y-y_i)^2) \ ,$$

and therefore,

$$\nabla\psi_i(x,y) = (8\pi)^{-1}(1+\log((x-x_i)^2+(y-y_i)^2)) \begin{bmatrix} (x-x_i) \\ (y-y_i) \end{bmatrix},$$

where here (x_i,y_i) is the ith data site.

The elements of M_1 are calculated numerically from line integrals as before. The calculation of these line integrals necessitates a large number of evaluations of the logarithm function and this slows the method down significantly, to the point at which the calculation of the matrix M_1 forms the largest part by far of the

calculation of a finite window spline. Obviously if the finite window method is to compete with the straightforward $\Omega \equiv \mathbf{R}^2$ type splines, it must not introduce vast amounts of extra computing time. For this reason a faster method based on the analytic evaluation of the line integrals was devised.

2.4. New routines for the evaluation of $A_\Omega(\varphi_i,\varphi_j)$ and $A_\Omega(\psi_i,\varphi_j)$.

The method proposed here is based on the fact that the φ functions are all in actual fact moderately low degree bivariate polynomials in Cartesian coordinates. The expressions for the φ functions as polynomials can be easily derived from the expressions for the functions c_j and s_j. These expressions are given by,

$$c_j(x,y) = \mathrm{Re}(x+iy)^j \text{ and } s_j(x,y) = \mathrm{Im}(x+iy)^j.$$

But we have

$$(x+iy)^j = \sum_{k=0}^{j} \binom{j}{k} x^{j-k}(iy)^k$$

So that

$$c_j(x,y) = \sum_{k=0}^{[j/2]} \binom{j}{2k} (-1)^k x^{j-2k} y^{2k}$$

and

$$s_j(x,y) = \sum_{k=0}^{[(j-1)/2]} \binom{j}{2k+1} (-1)^k x^{j-2k-1} y^{2k+1}$$

where here the square brackets indicate the integer part of the enclosed expression. In this way we can write down the polynomial expressions for all the φ functions.

We now proceed to the evaluation of the inner products $A_\Omega(\varphi_i,\varphi_j)$ for each i and j. Given the polynomial forms of the φ functions, we can see that each of these inner products becomes a sum of inner products of the form $A_\Omega(\varphi_i,x^n y^m)$. These inner products are,

$$\oint_{\partial\Omega} \nabla(\partial_n \varphi_i).\nabla x^n y^m - (\partial_n \nabla^2 \varphi_i) x^n y^m$$

and the integrand reduces to its first term when $i=1,2+4j$ or $3+4j$, since $\partial_n \nabla^2 \varphi_i \equiv 0$ in these cases.

In our case, of course, we are interested in $\partial\Omega$ which are piecewise linear and thus we have integrals to evaluate involving second and third derivatives of φ functions multiplied by simple polynomials of the form $x^n y^m$. From the expressions given for the derivatives of the φ functions in the previous section, we can see that integrals required for the evaluation of the inner product involving φ_{i+4j} are

expressible in terms of integrals of the forms,

$$\int c_{j+1}x^n y^m \ , \ \int s_{j+1}x^n y^m \ , \ \int r^2 c_{j-1}x^n y^m \ , \ \int r^2 s_{j-1}x^n y^m \ , \ \int c_j x^n y^m \text{ and } \int s_j x^n y^m \ .$$

Using the recurrence relations for c_{j+1} etc. given earlier and the relations,

$$r^2 c_{j-1} = xc_j + ys_j \text{ and } r^2 s_{j-1} = xs_j - yc_j \ ,$$

we can see that all of these integrals (and hence the inner product $A_\Omega(\varphi_{i+4j},\varphi_k)$) are equivalent to a linear combination of integrals of the form,

$$\int c_j x^n y^m \text{ and } \int s_j x^n y^m .$$

The final step is to notice that we can now express each of these integrals as a linear combination of integrals of the form, simply,

$$\int x^n y^m .$$

Before we proceed to the evaluation of these integrals let us first recap on the previous argument. This particular means of evaluating the inner products of φ functions was chosen for efficiency and ease of coding considerations in the programs. Thus the above argument is best expressed as an algorithm. This algorithm is the one actually used in the implementation.

for each linear piece Γ of the boundary $\partial\Omega$ **do**
  set up table of integrals $\int_\Gamma x^n y^m ds$
  for each j such that φ_{2+4j} is used **do**
    set up tables of integrals $\int_\Gamma c_j x^n y^m ds$ and $\int_\Gamma s_j x^n y^m ds$
    for each i such that φ_{i+4j} is used **do**
      for each $k \le i+4j$ **do**
        Find $\int_\Gamma \varphi_{i+4j}\varphi_k ds$
        Add to $A_\Omega(\varphi_k,\varphi_{i+4j})$

We now turn to the evaluation of the integrals $\int_\Gamma x(s)^n y(s)^m ds$ where Γ is one piece of the piecewise linear boundary $\partial\Omega$, and s is arc length. If the endpoints of Γ are (ξ_0,η_0) and (ξ_1,η_1), (going around the boundary in anticlockwise fashion), the the above integrals can be written,

$$\sqrt{((\xi_1-\xi_0)^2+(\eta_1-\eta_0)^2)}\int_0^1((1-\zeta)\xi_0+\zeta\xi_1)^n((1-\zeta)\eta_0+\zeta\eta_1)^m d\zeta \ .$$

The calculation of the integrals proceeds as follows. First define $I_{n,m}$ to be the integral in the above. If $n=0$ then we have,

$$I_{0,m} = \begin{cases} \eta_0^m & \text{if } \eta_0=\eta_1 \\ \dfrac{\eta_1^{m+1}-\eta_0^{m+1}}{(m+1)(\eta_1-\eta_0)} & \text{otherwise} \end{cases}$$

and similarly if $m=0$,

$$I_{n,0} = \begin{cases} \xi_0^n & \text{if } \xi_0=\xi_1 \\ \dfrac{\xi_1^{n+1}-\xi_0^{n+1}}{(n+1)(\xi_1-\xi_0)} & \text{otherwise.} \end{cases}$$

This leaves the cases where $n>0$ and $m>0$. We derive a recurrence formula for $I_{n,m}$ using integration by parts, so that we have,

$$I_{n,m} = \begin{cases} \xi_0^n I_{0,m} & \text{if } \xi_0=\xi_1 \\ \eta_0^m I_{n,0} & \text{if } \eta_0=\eta_1 \\ \dfrac{(\xi_1^n\eta_1^{m+1}-\xi_0^n\eta_0^{m+1})-n(\xi_1-\xi_0)I_{n-1,m+1}}{(m+1)(\eta_1-\eta_0)} & \text{otherwise.} \end{cases}$$

The case $\xi_0=\xi_1$ and $\eta_0=\eta_1$ should not normally occur as this means Γ is a single point and can therefore be eliminated from the boundary. Also notice that the last recurrence relation could be expressed in terms of $I_{n+1,m-1}$ and if the integrals were to be calculated individually it would make sense to use the version of the relation that led to the fewest recurrences. In our case however we in fact are calculating all the integrals for those n,m such that $n+m$ is less than a certain number, so that it is simplest to use one formula all the time, and this leads to no extra computation.

Let us now proceed to the evaluation of the inner products $A_\Omega(\psi_i,\varphi_j)$. Using a similar argument to those above this reduces to evaluating integrals of the form,

$$\oint_{\partial\Omega} \nabla(\partial_n\varphi_j).\nabla\psi_i - (\partial_n\nabla^2\varphi_j)\psi_i \ .$$

However we have, $\psi_i(x,y) = (16\pi)^{-1}r_i^2\log r_i^2$ where $r_i^2=(x-x_i)^2+(y-y_i)^2$ with $(x_i,y_i)=t_i$ the ith data site, so that,

$$\nabla\psi_i(x,y) = (8\pi)^{-1}\begin{bmatrix} x-x_i \\ y-y_i \end{bmatrix}(1+\log r_i^2) \ .$$

Using these results and an argument similar to before we can see that the inner product becomes a linear combination of integrals of the form,

$$\int_\Gamma x^n y^m \quad \text{and} \quad \int_\Gamma x^n y^m \log r_i^2 .$$

The first of these integrals we have already evaluated and it only remains to evaluate the integrals of the second form.

This is equivalent to evaluating the integrals,

$$\int_0^1 x(\zeta)^n y(\zeta)^m \log(a\zeta^2+2b\zeta+c)d\zeta$$

where $x(\zeta)=(1-\zeta)\xi_0+\zeta\xi_1$ and $y(\zeta)=(1-\zeta)\eta_0+\zeta\eta_1$ as before, and

$$\begin{aligned} a &= (\xi_1-\xi_0)^2+(\eta_1-\eta_0)^2 \\ b &= (\xi_1-\xi_0)(\xi_0-x_i)+(\eta_1-\eta_0)(\eta_0-y_i) \\ c &= (\xi_0-x_i)^2+(\eta_0-y_i)^2. \end{aligned}$$

Before we evaluate the above integrals we introduce the following related integrals,

$$J_{n,m} = \int_0^1 x(\zeta)^n y(\zeta)^m \frac{1}{(a\zeta^2+2b\zeta+c)} d\zeta$$

and

$$K_{n,m} = \int_0^1 x(\zeta)^n y(\zeta)^m \frac{\zeta}{(a\zeta^2+2b\zeta+c)} d\zeta.$$

These integrals can be evaluated using a set of reccurence relations in the same way that the $I_{n,m}$ were. The relations listed below are for the case $\xi_0 \neq \xi_1$ and $\eta_0 \neq \eta_1$, the other simpler cases being derived in similar fashion after removal of the constant factor as before.

For $n=0$ and $m=0$ we have,

$$J_{0,0} = \frac{1}{\sqrt{(ac-b^2)}} \left\{ \tan^{-1}\left[\frac{a+b}{\sqrt{(ac-b^2)}}\right] - \tan^{-1}\left[\frac{b}{\sqrt{(ac-b^2)}}\right] \right\}$$

and

$$K_{0,0} = \frac{1}{2a} \left\{ \log(a+2b+c) - \log(c) - 2bJ_{0,0} \right\}.$$

For $n=0$ and $m>0$ we have, by multiplying out the expression for one of the $y(\zeta)$'s $(=\eta_0+(\eta_1-\eta_0)\zeta)$,

$$J_{0,m} = \eta_0 J_{0,m-1} + (\eta_1-\eta_0)K_{0,m-1}$$

and

$$K_{0,m} = \eta_0 K_{0,m-1} + \frac{1}{a}(\eta_1-\eta_0)(1-2bK_{0,m-1}-cJ_{0,m-1})$$

where we have used the fact that,

$$\int_0^1 x(\zeta)^n y(\zeta)^m \frac{\zeta^2}{a\zeta^2+2b\zeta+c} d\zeta = \frac{1}{a}(1-2bK_{n,m}-cJ_{n,m}).$$

For $n>0$ we similarly have,

$$J_{n,m} = \xi_0 J_{n-1,m} + (\xi_1 - \xi_0) K_{n-1,m}$$

and

$$K_{n,m} = \xi_0 K_{n-1,m} + \frac{1}{a}(\xi_1 - \xi_0)(1 - 2bK_{n-1,m} - cJ_{n-1,m}).$$

Now return to the evaluation of the integrals,

$$L_{n,m} = \int_0^1 x(\zeta)^n y(\zeta)^m \log(a\zeta^2 + 2b\zeta + c) d\zeta.$$

Notice as with $I_{n,m}$ that if $\xi_0 = \xi_1$ we have $L_{n,m} = \xi_0^n L_{0,m}$, and similarly if $\eta_0 = \eta_1$. Otherwise we proceed as follows.

For $n=0$ and $m=0$ integration by parts, using 1 and $\log(a\zeta^2 + 2b\zeta + c)$ as the parts, gives,

$$L_{0,0} = \log(a+b+c) - 2(1 - bK_{0,0} - cJ_{0,0}).$$

For $n=0$ and $m>0$ we have, again by integrating by parts,

$$L_{0,m} = \frac{1}{(m+1)(\eta_1 - \eta_0)} \left\{ \eta_1^{m+1} \log(a+b+c) - \eta_0^{m+1} \log(c) - 2(aK_{0,m+1} + bJ_{0,m+1}) \right\}.$$

For $n>0$ we have,

$$L_{n,m} = \frac{1}{(m+1)(\eta_1 - \eta_0)} \left\{ \xi_1^n \eta_1^{m+1} \log(a+b+c) - \xi_0^n \eta_0^{m+1} \log(c) - n(\xi_1 - \xi_0) L_{n-1,m+1} - 2(aK_{n,m+1} + bJ_{n,m+1}) \right\}.$$

In this way all the inner products $A_\Omega(\psi_i, \varphi_j)$ can be found using the algorithm given below.

for each linear piece Γ of the boundary $\partial\Omega$ **do**
  set up table of integrals $\int_\Gamma x^n y^m ds$
  for each i in $1,\ldots,N$ **do**
    set up tables of integrals $L_{n,m} = \int_\Gamma x^n y^m \log r_i^2 ds$
    for each k such that φ_k is used **do**
      Find $\int_\Gamma \psi_i \varphi_k ds$
      Add to $A_\Omega(\varphi_k, \psi_i)$

Notice that the first part of this algorithm is the same as the first part of the algorithm used for finding the inner products of the φ functions. For this reason these algorithms can be interleaved to save some computation, and this is the

method used in our implementation. In the next section we will compare the speed of the two methods of evaluating all the inner products.

2.5. Comparison of the efficiency of the new with the old integration routines.

In order to assess the improvement in speed that the analytic integration routines give us over the numerical method of calculating the inner products two series of test programs are considered. The first of these series calculates the inner product matrices using the old and new routines for various numbers of points and extra (φ) functions. In these tests the region chosen was $\{-\frac{1}{2}\leq x,y\leq\frac{1}{2}\}$ and 25, 50 and 100 points are scattered randomly in this region. The inner product matrices $A_\Omega(\varphi_i,\psi_j)$ and $A_\Omega(\varphi_i,\varphi_j)$ are then calculated using both integration techniques. In the numerical case the integration tolerance was set to the value which gave agreement with the analytic routines to about 3 or 4 significant figures. The results presented in the following table are the CPU seconds used by each run on a Sun3/260 workstation using a floating point accelerator.

No. of Points	No. of Extra Functions			
	7		15	
	new	old	new	old
25	1.3	11.2	3.9	28.6
50	2.3	23.2	7.2	57.0
100	4.6	48.3	13.1	112.2

The figures above show that the new routines are very much quicker than the numerical routines. In fact when using 7 extra functions the analytic routines save about 90% of the processor time used by the numerical integration routines, regardless of the number of points defining ψ functions. When 15 extra functions are used the analytic routines are still approximately 85% faster. The reason for the slight decrease in advantage of the analytic routines probably lies in the fact that it takes more work to evaluate the integrals as the subscript of the φ functions increases, since they become higher order polynomials. There is also more work involved in evaluating the integrals numerically, as the degree of the φ polynomials increases, since the functions must be repeatedly evaluated as the integration proceedes, but it appears that this increase is proportionately less than the extra work incurred by the analytic routines. However for the range of the number of extra functions that we are concerned with the numerical routines will still be very significantly slower than analytic routines, probably by at least 80%.

Although the above figures appear dramatic, they are not really a fair test of the new routines since there are still several other calculations to perform after obtaining the inner product matrices before the spline is known. These calculations remain common to both the analytic and numerical methods used above. In order to obtain a fair comparison we should include these calculations since they may swamp the timings given above.

For this reason a second series of test programs were run and results obtained. The programs calculate all the coefficients of the basis functions for the splines corresponding to the cases given above, and also of the thin plate spline for comparison. The timings are in the following table.

No. of Points	No. of extra functions				
	0	7		15	
		new	old	new	old
25	0.5	1.8	11.3	4.2	28.4
50	4.4	6.9	27.3	11.4	60.4
100	36.4	41.4	83.0	50.8	147.2

Notice now that some of the comparisons are much less dramatic. When using 7 extra functions the improvement in speed ranges from about 84% to as low as 50% and when using 15 extra functions the improvement is better, ranging from 85% to 65%. In both cases the lower of these figures occurs when the largest number of points is used and the highest when 25 points are being used. This seems to suggest that the larger the number of points the more the improvement in speed of the analytic routines is swamped out by other considerations. This is consistent with the fact that both methods are now calculating the inverse of an $N-3 \times N-3$ matrix where N is the number of points concerned and this is in fact an $O(N^3)$ operation.

In this way it can be seen that as N increases the calculation of this inverse (along with other calculations) begins to dominate the calculation of the spline. However the analytic routines still have a significant improvement over the numerical routines for all practical values of N. (For a discussion of what are practical values for N see elsewhere in this thesis and Wahba (1979)).

Chapter 3.
Fitting Finite Window Splines to Simulated and Real Data.

The results that we present fall into three categories, namely reproduction of Dyn and Levin's results, further simulation studies and application to real data. Firstly we consider the reproduction of Dyn and Levin's results, which are based on the interpolation of three functions in an L-shaped domain. Our own simulation study concentrates on an area not reported by Dyn and Levin, namely the choice of region Ω over which to fit the spline. Lastly we show the procedure in action on geological data taken from O'Connor and Leach (1979).

3.1. The Results of Dyn and Levin.

In Dyn and Levin (1982) the authors present the results of a simulation study of the finite window splines. They use an L-shaped region Ω as below, and generate data at randomly generated points in this domain, from three functions f_1, f_2 and f_3, also shown below. The simulation was carried out for 13 and 27 points in Ω and for the values of the *closeness of approximation* parameter n (the number terms in the sequence of φ functions), of 0. 7 and 15. The table shows the roughness of the solution $\hat{u}$ for each case.

$$\Omega = \{ (x,y) \mid -0.5 \le x,y \le 0.5, x \ge 0 \text{ or } y \ge 0 \}.$$

$$f_1 = 4xy,\ f_2 = \sin(10xy),$$

$$f_3 = \exp(-25(x^2+y^2-0.1)^2).$$

Table 3.1.1.
Roughness over Ω of Dyn and Levin's Fitted Spline for Three Test Functions.

No. of Points	No. of φ fns.	f_1	f_2	f_3
13	0	25.0	98.0	85.0
	7	17.4	76.0	78.0
	15	16.6	72.0	74.0
27	0	28.1	105.0	132.0
	7	19.0	87.0	124.0
	15	18.3	82.0	119.0

The authors do not specify, in their paper, the location of the data sites within the L-shape. Thus we cannot expect to reproduce their results exactly. For this reason we have repeated their simulations with ten replications (using different sets of data sites) and we present the means and standard deviations of these results. (See table below).

Table 3.1.2.
Mean Roughnesses over Ω of Fitted Splines
(with Standard Deviations, 10 Replications).

No. of Points	No. of φ fns.	f_1	f_2	f_3
13	0	21.9(4.7)	88.9(15.7)	71.3(24.0)
	7	15.6(2.8)	72.7(13.7)	64.2(21.4)
	15	14.9(2.8)	67.6(11.8)	62.1(21.0)
27	0	28.3(1.9)	121.9(12.7)	148.8(16.3)
	7	19.6(1.0)	107.8(15.3)	139.7(17.0)
	15	19.0(1.1)	97.8(12.0)	137.4(16.5)

It can be seen from the tables that all of Dyn and Levin's results lie within two standard deviations of our mean results, and thus we conclude that we are producing consistent results. Since the pattern of the results presented by the authors is also followed by our results, the conclusions in their paper apply equally well here. The authors also indicate that away from the boundary of Ω the finite window interpolants recover the original function more accurately than the straight forward thin plate spline, and this result is also supported by our study.

3.2. The Choice of Region Ω Over Which to Fit the Finite Window Splines.

In this section we present the results of a study designed to help determine the best choice of region over which to fit finite window splines. This is an important question when the data is presented as a list of data sites and interpolation values, but may be less important when it is known that the data sites were chosen in order to cover a particular area. For instance, when mapping an area, an investigator might attempt to spread his recording points evenly over the area to be mapped. In this case the choice of region, Ω, is natural and it is to be hoped that imperfections in the spread of points does not adversely affect the method. It can be seen however that an imperfection in the spread of points is equivalent to covering some other region, and it is the sensitivity of the method to the choice of

region Ω that concerns us here.

There is another reason why a certain amount of insensitivity in the method to choice of region is desirable. Suppose the analyst is given only the data sites and values at those sites without any associated region. In this case the analyst must choose an appropriate region in which to apply the method. In order to do this effectively he must know what is to be gained by using a finite window, over taking $\Omega \equiv \mathbf{R}^2$, and how sensitive to the choice of this essentially *arbitrary* region the method is.

To help answer these questions or at least allay some of the fears they may cause, and also to show the method in use we have conducted two studies called Sizetest and Shapetest.

3.2.1. Sizetest.

The purpose of this simulation study is to get some indication as to whether the size of the region enclosing the data sites has a significant effect on two properties of the interpolant. The properties considered are the residual roughness and the "goodness" of recovery of a sample function by the interpolant.

For our study we chose to use five functions from which to simulate data. Data was simulated from these functions using 25 and 50 points uniformly spread over the square $\{-0.499 \le x,y \le 0.499\}$. Interpolants were then constructed using no extra functions (that is Thin plate splines) and also using 7 and 15 extra functions over the regions $\{-0.5 \le x,y \le 0.5\}$, $\{-0.6 \le x,y \le 0.6\}$ and the region $\{-1.0 \le x,y \le 1.0\}$, over which all the surfaces are contoured. For each interpolant generated the residual roughnesses and approximations to $\int (f-\hat{f})^2/\text{area}$, $\int (f_x-\hat{f}_x)^2/\text{area}$ and $\int (f_y-\hat{f}_y)^2/\text{area}$ over each of the three regions, were calculated and the results tabulated. We introduce the following notation to make the tables clearer.

$$\Omega_1 = \{ -0.5 \le x,y \le 0.5 \}$$

$$\Omega_2 = \{ -0.6 \le x,y \le 0.6 \}$$

$$\Omega_3 = \{ -1.0 \le x,y \le 1.0 \}$$

The first function chosen was $4xy$, as used by Dyn and Levin. A quadratic is a sensible first choice, since they are the first polynomials not recovered exactly by spline interpolants, whereas the first degree functions are recovered exactly, and this quadratic maintains a point of contact with Dyn and Levin's work. This function is also one of the extra basis functions (φ functions) used in finite

window splines. The roughnesses of the fitted splines can been seen in table 3.2.1.1. When looking at the tables of roughnesses which follow, it must be remembered that the numbers in each column are integrals over the regions, so that these numbers are not really comparable along the rows. This problem may be alleviated by standardising the integrals to give roughness per unit area, but this does not solve the problem since the splines have quite different relationships to each of these regions. The data is only scattered in the smallest region, so comparison of the roughness over this region with the roughnesses over the larger regions gives us some idea of the roughness caused by edge effects.

In the table we expect the smallest roughness over a given region to occur when the spline is fitted over the same region, and when the higher number of extra functions is used. This is in fact the case in both the twenty-five and the fifty point interpolants. Notice also that for the 25 point interpolants and for each region Ω_1, Ω_2 and Ω_3 the roughness of the spline fitted over that region is considerably less than the roughness over the same region of the thin plate spline. The splines over Ω_1 and Ω_2 have reductions respectively of 36.8 to 26.8 and 45.8 to 33.6, which are reductions of more than 25%. The spline over Ω_3 has a more modest reduction in roughness of 63.1 to 52.2, about 15%. Similar results apply to the 50 point case. This is entirely consistent with the Dyn and Levin results mentioned above.

The splines fitted over the regions Ω_1 and Ω_2 have very close roughnesses over both these regions. This suggests that the choice of boundary does not make that much difference to the roughness of the solution spline provided it is close to the data. However the spline fitted over the smallest region Ω_1 actually has a larger roughness than the thin plate spline, when measured over the largest region Ω_3. This seems to confirm the intuitive idea that the finite window spline is transferring roughness out of its window and into the rest of $\mathbf{R}^2$.

Over the whole table the roughnesses of the 7 φ function spline and the 15 φ function spline are very close, except possibly in the cases of splines over Ω_1 and Ω_2 where the roughnesses are measured over the region Ω_3. Again the results suggest that splines increase the roughness outside their regions quite considerably in order to decrease the roughness inside their region by a relatively small amount.

Table 3.2.1.2. gives an approximation to the integrated squared difference between interpolating splines and the original function per unit area. The figures in this and the following table 3.2.1.3. regarding gradient error, tell roughly the same story as the the roughness figures. All the finite window splines are an improvement on the thin plate spline in terms of recovery error and again there is

Table 3.2.1.1.

Roughnesses of Splines Fitted to f_1.

No. of Points	Spline fitted over		Residual Roughness over Ω_1	Ω_2	Ω_3
25		R^2	36.8	45.8	63.1
	Ω_1	7 φ fns.	26.8	34.4	72.6
		15 φ fns.	26.6	33.8	69.0
	Ω_2	7 φ fns.	26.8	33.6	63.6
		15 φ fns.	26.8	33.2	57.1
	Ω_3	7 φ fns.	29.2	35.5	52.2
		15 φ fns.	28.9	35.1	51.5
50		R^2	41.6	54.6	79.1
	Ω_1	7 φ fns.	29.7	39.0	84.3
		15 φ fns.	29.6	39.1	110.7
	Ω_2	7 φ fns.	29.9	38.4	75.7
		15 φ fns.	29.9	38.2	72.2
	Ω_3	7 φ fns.	32.5	40.8	63.3
		15 φ fns.	32.3	40.6	62.5

Table 3.2.1.2.

Value Recovery Error for f_1.

No. of Points	Spline fitted over		Error in Value per unit area, over Ω_1	Ω_2	Ω_3
25		R^2	0.0046	0.0259	0.8321
	Ω_1	7 φ fns.	0.0008	0.0040	0.1146
		15 φ fns.	0.0010	0.0053	0.1884
	Ω_2	7 φ fns.	0.0011	0.0060	0.1752
		15 φ fns.	0.0013	0.0071	0.2707
	Ω_3	7 φ fns.	0.0024	0.0132	0.4130
		15 φ fns.	0.0023	0.0126	0.4213
50		R^2	0.0010	0.0114	0.6608
	Ω_1	7 φ fns.	0.0001	0.0011	0.0634
		15 φ fns.	0.0001	0.0012	0.0779
	Ω_2	7 φ fns.	0.0002	0.0018	0.1006
		15 φ fns.	0.0002	0.0021	0.1496
	Ω_3	7 φ fns.	0.0004	0.0047	0.2708
		15 φ fns.	0.0004	0.0045	0.2835

Table 3.2.1.3.

Gradient Recovery Error for f_1.

No. of Points	Spline fitted over		Error in x-derivative per unit area, over Ω_1	Ω_2	Ω_3	Error in y-derivative per unit area, over Ω_1	Ω_2	Ω_3
25		R^2	0.288	0.671	3.549	0.189	0.551	3.459
	Ω_1	7 φ fns.	0.046	0.100	0.487	0.029	0.080	0.499
		15 φ fns.	0.065	0.155	1.205	0.031	0.088	0.701
	Ω_2	7 φ fns.	0.068	0.151	0.750	0.043	0.119	0.734
		15 φ fns.	0.082	0.198	1.495	0.046	0.136	1.135
	Ω_3	7 φ fns.	0.149	0.342	1.766	0.094	0.273	1.703
		15 φ fns.	0.142	0.330	1.885	0.089	0.262	1.794
50		R^2	0.114	0.417	3.164	0.106	0.388	3.080
	Ω_1	7 φ fns.	0.011	0.039	0.300	0.010	0.037	0.293
		15 φ fns.	0.012	0.044	0.443	0.011	0.039	0.460
	Ω_2	7 φ fns.	0.017	0.063	0.476	0.016	0.060	0.465
		15 φ fns.	0.021	0.079	0.837	0.019	0.071	0.756
	Ω_3	7 φ fns.	0.047	0.170	1.291	0.044	0.160	1.260
		15 φ fns.	0.045	0.166	1.419	0.042	0.155	1.376

little difference between the splines over Ω_1 and Ω_2. The best figures represent a reduction in recovery error by the finite window splines over the thin plate splines of around 90%, and the gradient figures are no less dramatic.

We can gain more information about the differences in the fitted splines by looking at the following figures. These figures are contour maps of the functions drawn over the region $\{-1 \leq x,y \leq +1\}$ using the contouring package CONICON3 developed at Bath University by Prof. R Sibson. On the contour maps the small plus symbols mark the positions of the data sites used and the boundary of the region over which the spline is fitted is marked for finite window splines, where it differs from the whole contouring area.

The contouring procedure uses a piecewise quadratic C^1 interpolant, which it actually contours, so there is a second level of interpolation error induced by contouring this approximant. This interpolant is based on value and gradient information supplied at a grid of 21×21 points and thus we expect this secondary error to be very small. In fact contour maps drawn using information at an 11×11 grid are not visibly different from those presented here, and in any case the contouring interpolation error is known to be high only when the contoured function is extremely non-quadratic in form or the function is very flat.

Figure 3.2.1.1 shows the actual function $4xy$ that the subsequent splines are attempting to recover. Figure 3.2.1.2 shows the 25 point interpolant with $\Omega=\mathbf{R}^2$. Within the body of the data the interpolation is not as bad as may appear from this contour map. However towards the edge of the data, the contours bend around significantly in order that the spline is *smooth* in the corners of the plot.

In figures 3.2.1.3 and 3.2.1.4, 7 extra functions have been used to fit the spline over the regions Ω_1 and Ω_2 respectively. The visible improvement in this case is striking, but it must be remembered that most of this improvement occurs a long way from the data. This is therefore to be expected, since the data itself effectively *ties down* the interpolant in its vicinity but it is the *smoothness* criteria which dominates the form of the spline away from the data.

The spline in figure 3.2.1.5 is the 25 point, 7 extra function over Ω_3 interpolant. Even this spline is a visible improvement over the thin plate spline, but it is less so than figures 3.2.1.3 and 3.2.1.4. The 15 extra function and 50 point spline interpolants show similar features, to those above and figure 3.2.1.6 is the 50 point, 7 extra function interpolant over Ω_1, the spline with lowest recovery error of all those fitted.

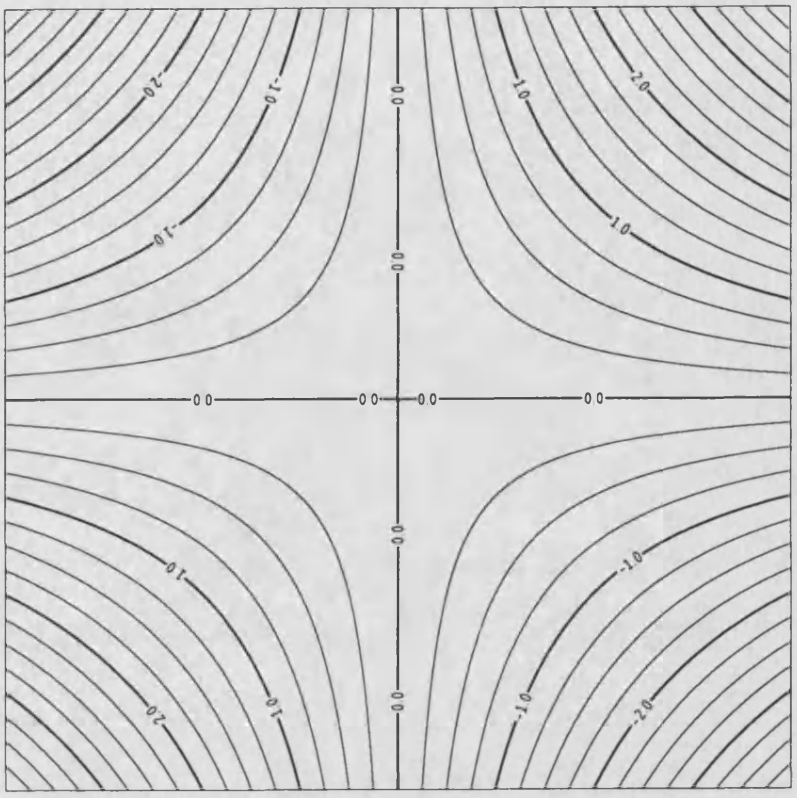


fig 3.2.1.1. Function 1

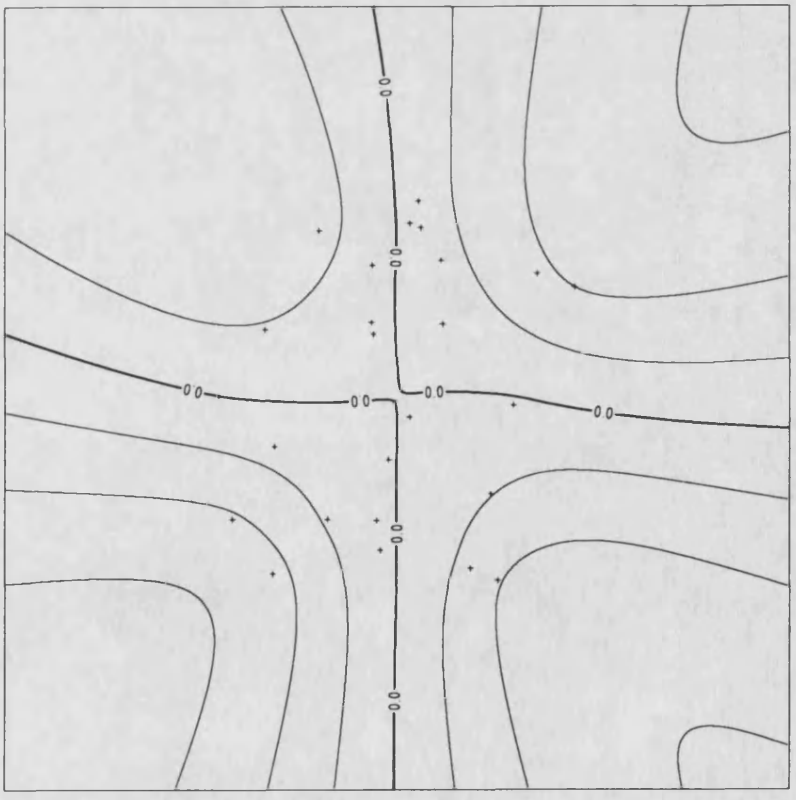


fig 3.2.1.2. 25 points, 0 extra functions

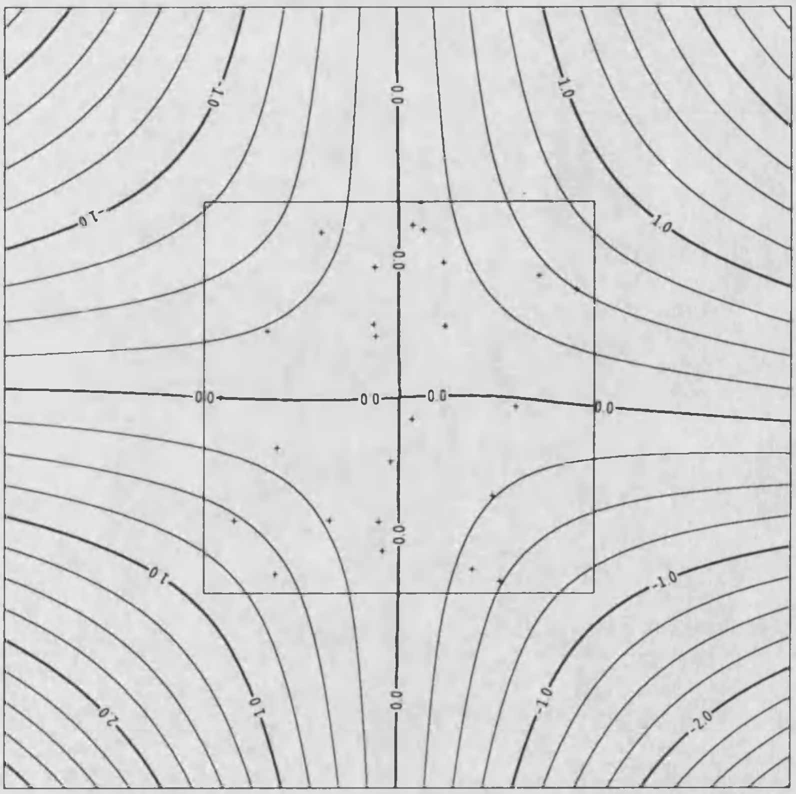


fig 3.2.1.3. 25 points, 7 extra functions, region 1

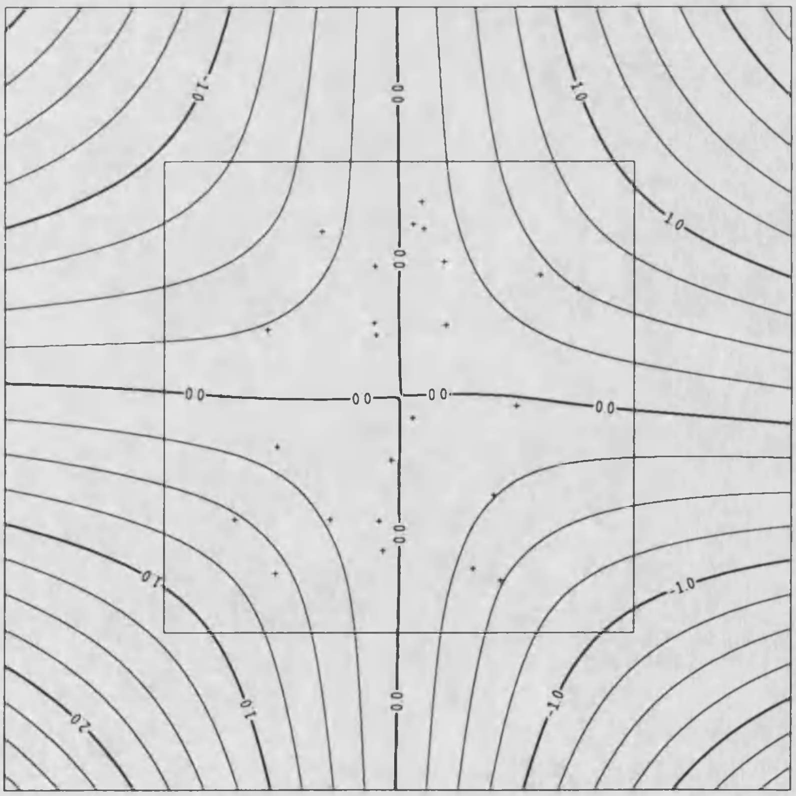


fig 3.2.1.4. 25 points, 7 extra functions, region 2

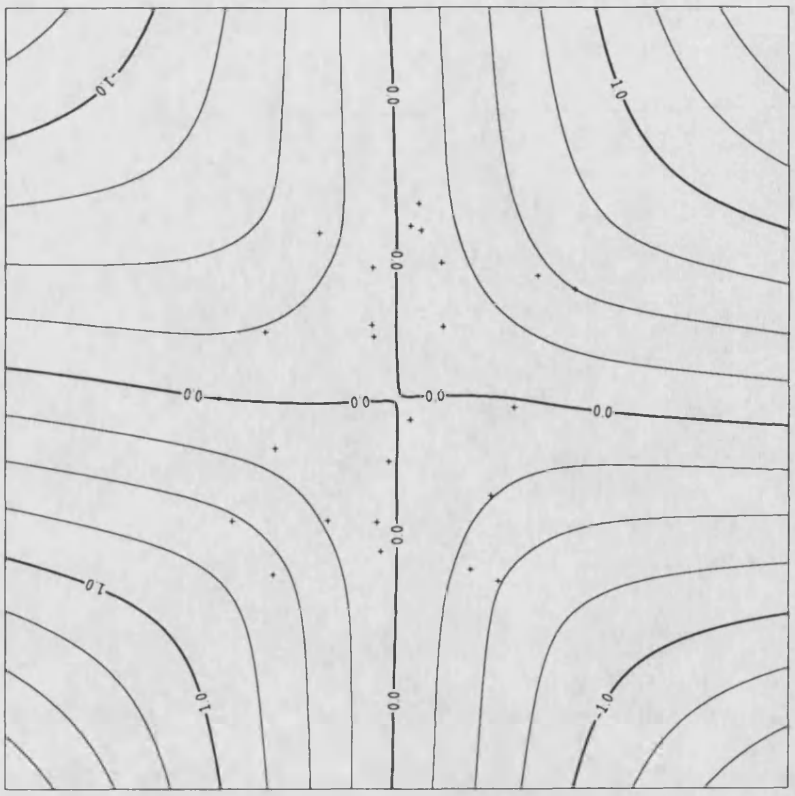


fig 3.2.1.5. 25 points, 7 extra functions, region 3

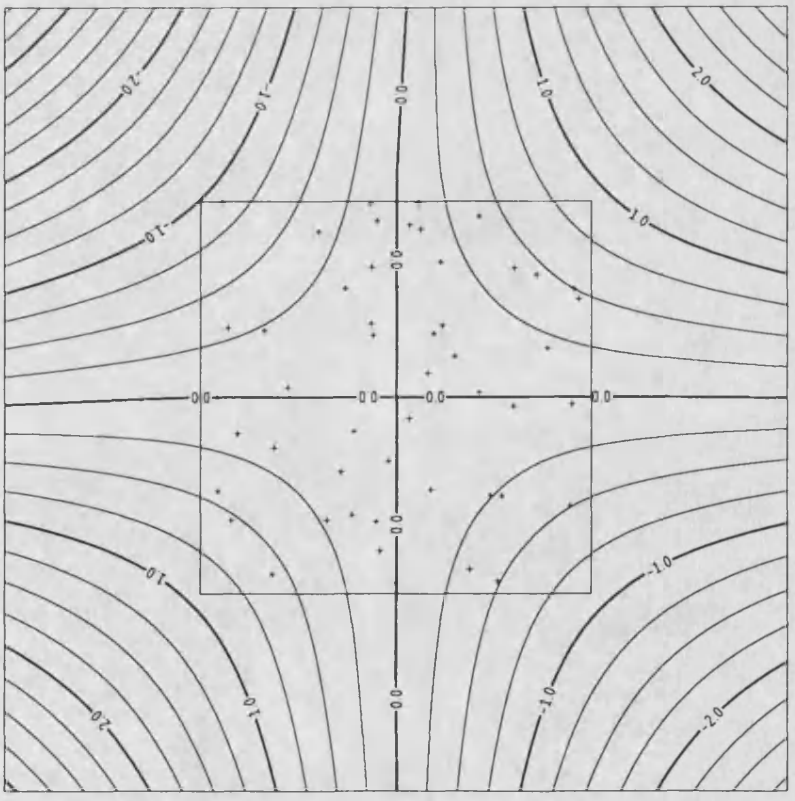


fig 3.2.1.6. 50 points, 7 extra functions, region 1

The second function chosen as a test function for the Sizetest procedure was,

$$f_2(x,y) = 3\left[(x-0.3)^3-3(x-0.3)(y+0.2)^2\right]+\left[3(x-0.3)^2(y+0.2)-(y+0.2)^3\right].$$

This is in fact a linear combination of two of the φ functions, namely φ_6 and φ_7. This cubic function was chosen as the next most complicated polynomial after a quadratic, and it is interesting to see what happens to the results given in the previous section as the function being recovered becomes more complicated.

This cubic also has that property that it satisfies one of the boundary conditions that a spline satisfies but not the others. Recall that the spline satisfies the boundary conditions,

$$\nabla\frac{\partial\hat{u}}{\partial n}=\mathbf{0} \text{ and } \frac{\partial}{\partial n}\nabla^2\hat{u}=0 \quad \text{on } \partial\Omega.$$

The previous function $4xy$ had the property that it satisfied two out of three of these boundary conditions itself, at least in our case. The cubic function above however only satisfies the third of these conditions (at least when Ω is a square), and so we may expect that a spline has more difficulty recovering the edge behaviour of this function. The next function we shall consider satisfies none of these conditions (in our case) and so the edge effects may be even more substantial.

Table 3.2.1.4. presents the residual roughness values over the same regions as before of the same set of interpolants as before. These figures represent much the same reductions as the figures for f_1 with one exception. The roughnesses over Ω_1 of the splines fitted over Ω_1 and Ω_2 with 7 extra functions are, 141.6 and 140.8 respectively. We would expect the first of these figures to be lower than the second since this first spline was fitted over the region the roughness was actually measured over, and should therefore be the *smoothest* interpolant possible over this region. The spline fitted over the slightly larger region has a roughness which is lower by less than 1% and this is due to the fact that 7 extra functions is probably too few extra functions to take to obtain the expected results. It must be remembered that all the finite window splines presented here are not actually true splines, rather they are approximations to the minimal roughness splines fitted over the regions in question, and the number of extra functions controls the closeness of this approximation.

Table 3.2.1.5. has the figures for value recovery error of the interpolants. As can be seen from the table the figures show much the same pattern as with the previous function, although the overall level of error is considerably higher. One interesting result however can be seen in the recovery errors over Ω_3 in the 25

Table 3.2.1.4.

Roughnesses for f_2.

No. of Points	Spline fitted over		Residual Roughness over Ω_1	Ω_2	Ω_3
25		R^2	185.5	226.8	296.8
	Ω_1	7 φ fns.	141.6	194.3	661.7
		15 φ fns.	137.9	178.7	1199.1
	Ω_2	7 φ fns.	140.8	178.5	415.7
		15 φ fns.	139.4	173.9	472.2
	Ω_3	7 φ fns.	160.5	193.2	262.5
		15 φ fns.	156.6	188.9	258.8
50		R^2	243.7	333.5	475.9
	Ω_1	7 φ fns.	176.8	259.4	1045.7
		15 φ fns.	173.9	241.9	792.0
	Ω_2	7 φ fns.	177.1	241.8	705.9
		15 φ fns.	176.7	236.4	603.1
	Ω_3	7 φ fns.	206.8	271.8	408.4
		15 φ fns.	201.3	264.3	401.8

Table 3.2.1.5.

Value Recovery Error for f_2.

No. of Points	Spline fitted over		Error in Value per unit area, over Ω_1	Ω_2	Ω_3
25		R^2	0.0184	0.1070	4.2001
	Ω_1	7 φ fns.	0.0045	0.0253	0.9852
		15 φ fns.	0.0055	0.0358	3.2651
	Ω_2	7 φ fns.	0.0076	0.0429	1.7025
		15 φ fns.	0.0074	0.0446	2.7586
	Ω_3	7 φ fns.	0.0143	0.0814	3.2483
		15 φ fns.	0.0135	0.0767	3.1646
50		R^2	0.0029	0.0311	2.9952
	Ω_1	7 φ fns.	0.0003	0.0040	0.3584
		15 φ fns.	0.0006	0.0069	1.2763
	Ω_2	7 φ fns.	0.0007	0.0078	0.7291
		15 φ fns.	0.0009	0.0100	1.5472
	Ω_3	7 φ fns.	0.0019	0.0206	2.0442
		15 φ fns.	0.0017	0.0190	1.9921

point case. Here the 15 extra function interpolant fitted over Ω_1 has an extremely large recovery error compared to the trend in the column and to the 7 extra function spline. We shall discuss this later in the context of the contour maps of the interpolants.

Figure 3.2.1.7 shows the theoretical function that we are trying to recover, and figure 3.2.1.8 shows the 25 point thin plate spline interpolant. Immediately we see that the recovery is much worse than in the case of the previous function. We believe this largely due to the extremely flat nature of the function around the bottom right of the region Ω_1. This flatness where the data influencing the whole lower right quadrant of the contoured area is situated makes it difficult to predict the behaviour of the surface outside of Ω_1. Of course this kind of extrapolation is not to be recommended, but even so, since the thin plate spline is smoothing in this area, it may propagate an edge effect right up to the edge of the data. This becomes more apparent when we compare figure 3.2.1.8 with figure 3.2.1.9 the interpolant fitted over Ω_1 with 7 extra functions. The whole shape of the interpolant is different throughout the whole of Ω_3 the contoured region, and the edge effects now recognised in the thin plate spline, are seen to propagate right up to the edge of the data. Well within the data however the interpolation conditions take over from the smoothness as the major determining factor, and the differences between figures 3.2.1.8 and 3.2.1.9 are minimal. If we were smoothing rather than interpolating we may have found the edge effects propagating further into the data, since then we do not enforce interpolation, only closeness to the data. We shall discuss this aspect in a later section.

Figure 3.2.1.10 shows the spline over Ω_1 with 25 points and 15 extra functions. This is the spline with the anomalous value recovery error in the table above. In the lower right hand corner of the contoured region it appears to be doing better than either of the previous splines, but down the left hand side of the plot it is exhibiting a kind of *over correction.* In pushing roughness out of the region of the data it has made this left hand area too rough, whereas the thin plate spline makes it too smooth. This may be an indication that small numbers of extra functions can do better than larger, and accounts for the poor performance of the 15 extra function spline, over the region Ω_3 as a whole. Looking again at the table of roughnesses and recovery errors, we see that the spline fitted over Ω_2 is most stable in this respect, that is the variation in the results for the 7 and 15 extra function splines is least when the spline is fitted over this region.

As mentioned earlier the third function used in the simulations, does not satisfy any of the boundary conditions mentioned above. The function is still quite simple however (namely cubic), and a further quartic function has been used in

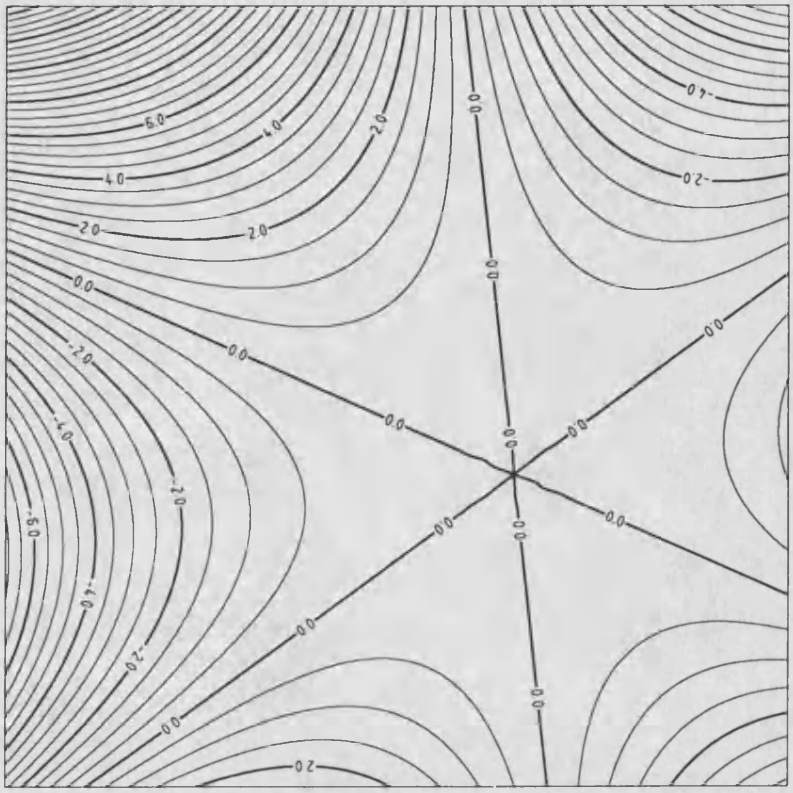


fig 3.2.1.7. Function 2

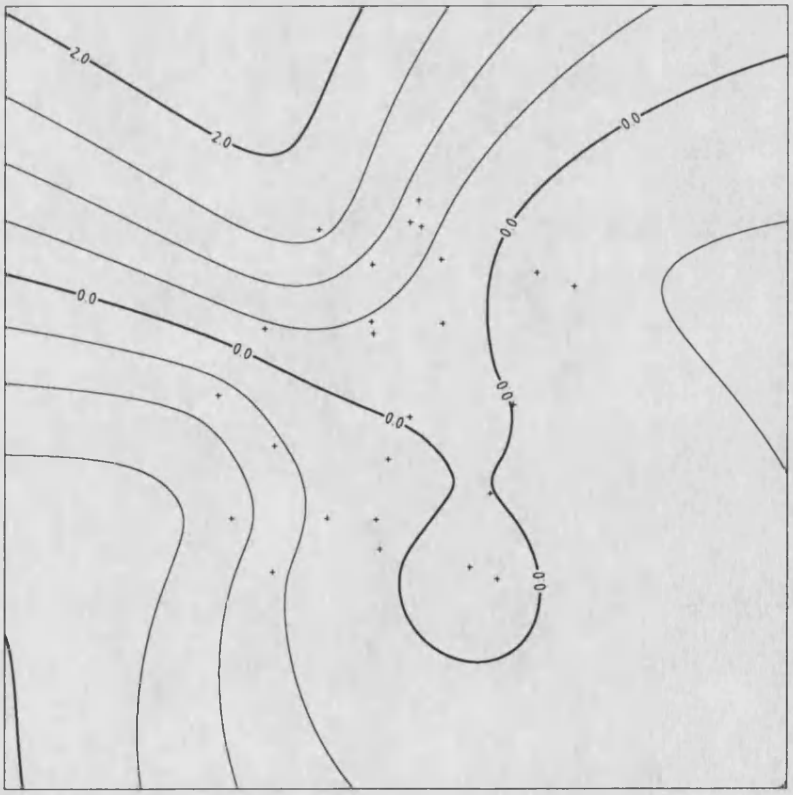


fig 3.2.1.8. 25 points, 0 extra functions

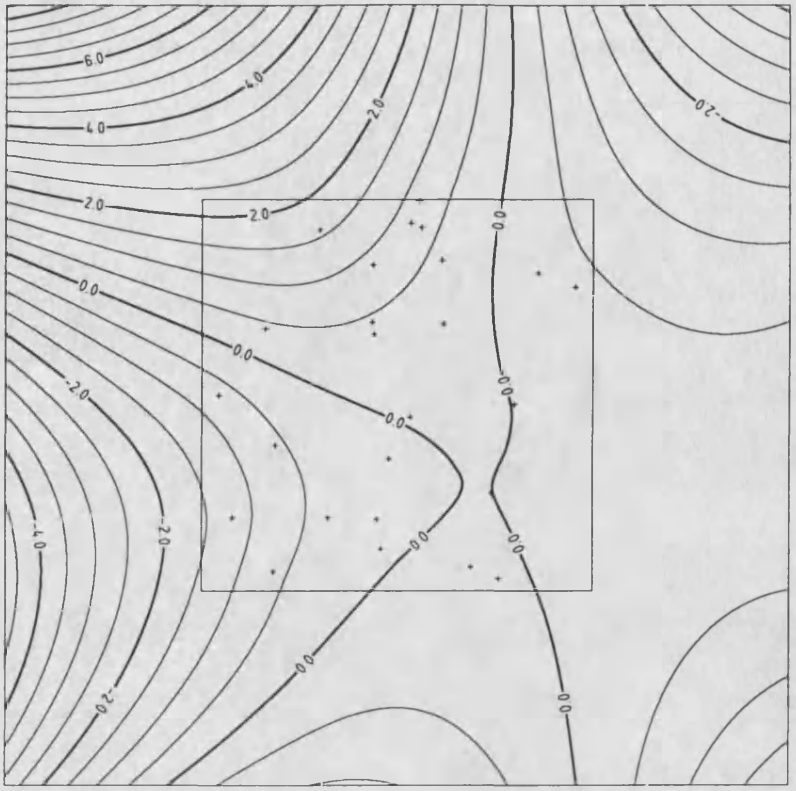


fig 3.2.1.9. 25 points, 7 extra functions, region 1

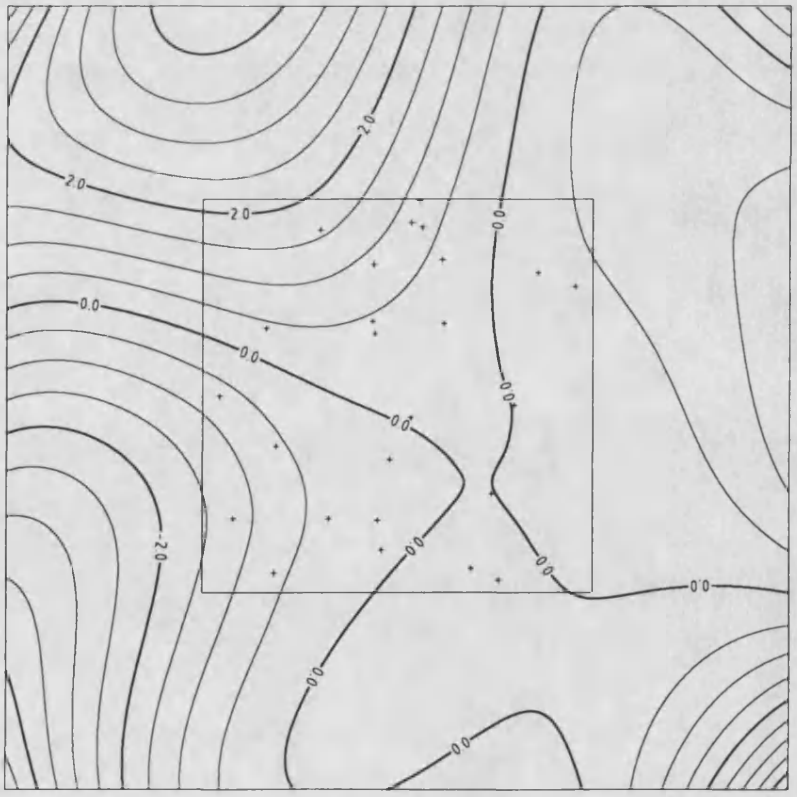


fig 3.2.1.10. 25 points, 15 extra functions, region 1

the simulations, in order to compare these effects of function simplicity and the satisfying of boundary conditions. These functions are,

$$f_3(x,y) = 4\left[x^3+x(y+0.2)^2\right]+\left[x^2(y+0.2)+(y+0.2)^3\right]$$

and

$$f_4(x,y) = 3(x-0.1)^4-2y^2.$$

The tables of roughnesses and value recovery error show similar patterns to those in the previous examples but with all effects less pronounced.

A final simulation was carried out using a function of an entirely different nature to the polynomials used previously. This function, f_5, is given by,

$$f_5 = 1.7\exp\left[-((x+0.2)^2+\tfrac{1}{2}y^2)/0.08\right]-2\exp\left[-((x-0.1)^2+(y-0.05)^2)/0.18\right]$$

The results are tabulated in tables 3.2.1.6. and 3.2.1.7.

As the theory indicates all the roughnesses can be reduced by fitting finite window splines instead of thin plate splines, although the reductions are all slight. When a finite window spline is fitted the recovery error is often actually worse than that for the thin plate spline interpolant to f_5.

Figure 3.2.1.11 shows the theoretical function f_5 and figures 3.2.1.12, and 3.2.1.13 show the thin plate and 15 extra function over Ω_2, 50 point interpolant splines respectively. Both splines have a low recovery error close to the data, where the interpolation conditions have most influence, but neither can recover the rapid tail out to zero away from the data, and the finite window spline is in fact worse in this area. The thin plate spline does better away from the data because it at least assumes smoothness there whereas, the finite window spline does not, but neither allows sufficient roughness at the edge of the area covered by the data for the spline to tail out to zero.

3.2.2. Shapetest.

Motivated by the results obtained when repeating Dyn and Levin's simulations, we decided to see if a change in shape of the region Ω would affect the solution spline. An important question to be asked here is should the region be convex. Clearly, in the definition of the spline problem, it is necessary only that the region, be such that we can integrate over it. In fact the same definition could be used if Ω had more than one component. In the mathematical solution, it is important only that the region Ω be one in which a restricted version of the divergence theorem, namely bivariate integration by parts, applies. It can be seen that the derivation of the unique solution spline could go through, even if Ω had a

Table 3.2.1.6.

Roughness for f_5.

No. of Points	Spline fitted over		Residual Roughness over Ω_1	Ω_2	Ω_3
25		R^2	375.4	399.4	420.1
	Ω_1	7 φ fns.	362.4	394.4	586.2
		15 φ fns.	359.6	399.8	1211.9
	Ω_2	7 φ fns.	364.1	389.1	466.4
		15 φ fns.	362.0	387.3	519.5
	Ω_3	7 φ fns.	372.3	395.3	417.0
		15 φ fns.	371.5	394.5	416.5
50		R^2	482.1	504.7	524.4
	Ω_1	7 φ fns.	477.0	503.8	643.2
		15 φ fns.	476.6	506.3	1044.6
	Ω_2	7 φ fns.	477.5	499.7	552.0
		15 φ fns.	477.2	499.4	596.4
	Ω_3	7 φ fns.	480.9	502.9	522.9
		15 φ fns.	480.9	502.9	522.8

Table 3.2.1.7

Value Recovery Error for f_5.

No. of Points	Spline fitted over		Difference in Value(/area) over Ω_1	Ω_2	Ω_3
25		R^2	0.0115	0.0192	0.0392
	Ω_1	7 φ fns.	0.0162	0.0319	0.6497
		15 φ fns.	0.0195	0.0408	1.0664
	Ω_2	7 φ fns.	0.0147	0.0273	0.2918
		15 φ fns.	0.0163	0.0311	0.3324
	Ω_3	7 φ fns.	0.0122	0.0211	0.0739
		15 φ fns.	0.0126	0.0221	0.0804
50		R^2	0.0002	0.0005	0.0399
	Ω_1	7 φ fns.	0.0003	0.0039	0.6079
		15 φ fns.	0.0004	0.0053	1.0744
	Ω_2	7 φ fns.	0.0002	0.0019	0.2754
		15 φ fns.	0.0003	0.0021	0.3481
	Ω_3	7 φ fns.	0.0002	0.0007	0.0696
		15 φ fns.	0.0002	0.0007	0.0699

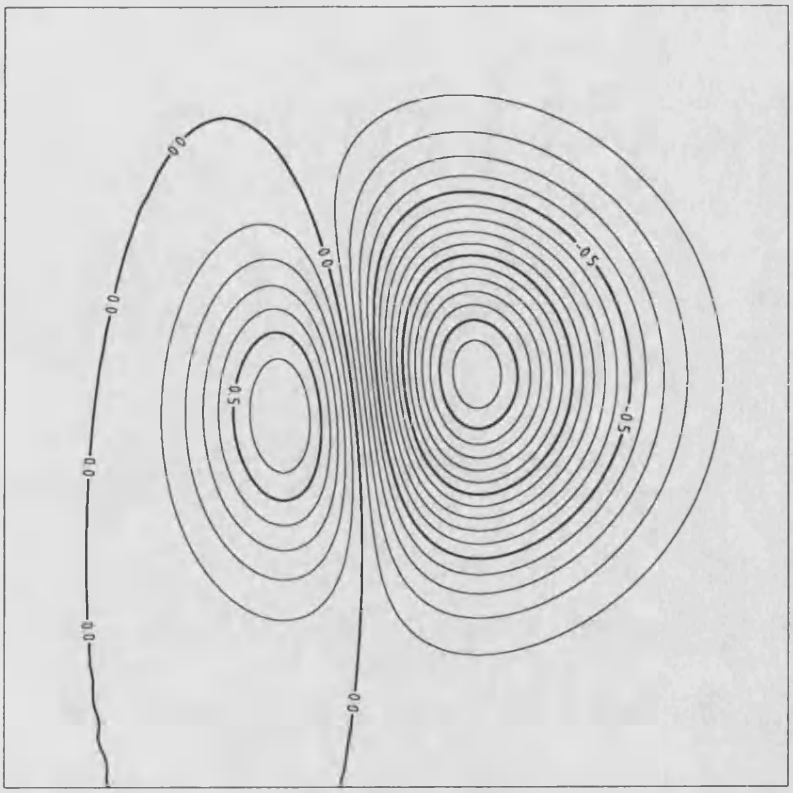


fig 3.2.1.11. Function 5

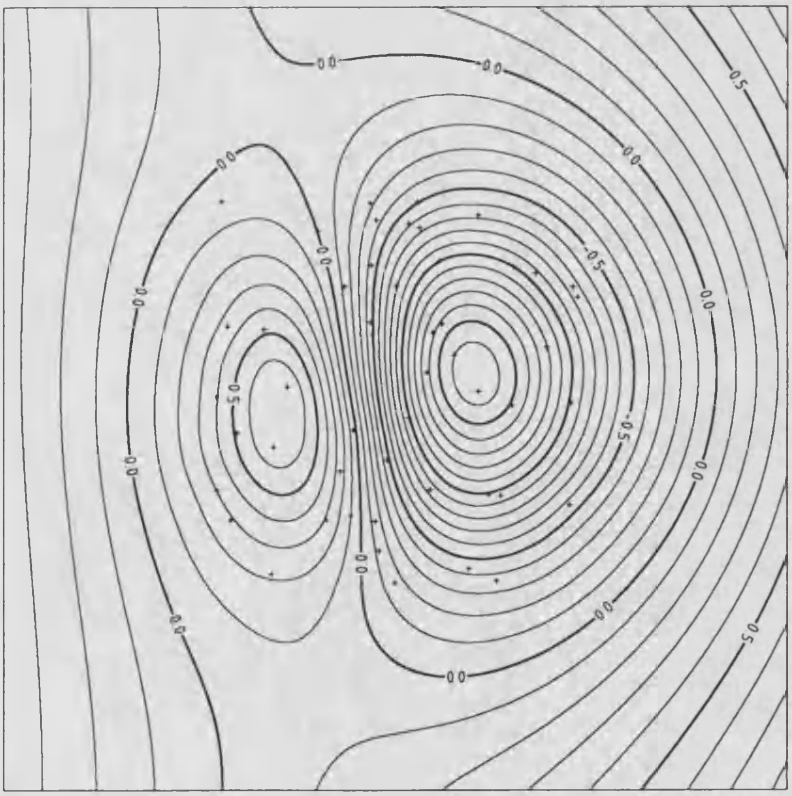


fig 3.2.1.12. 50 points, 0 extra functions

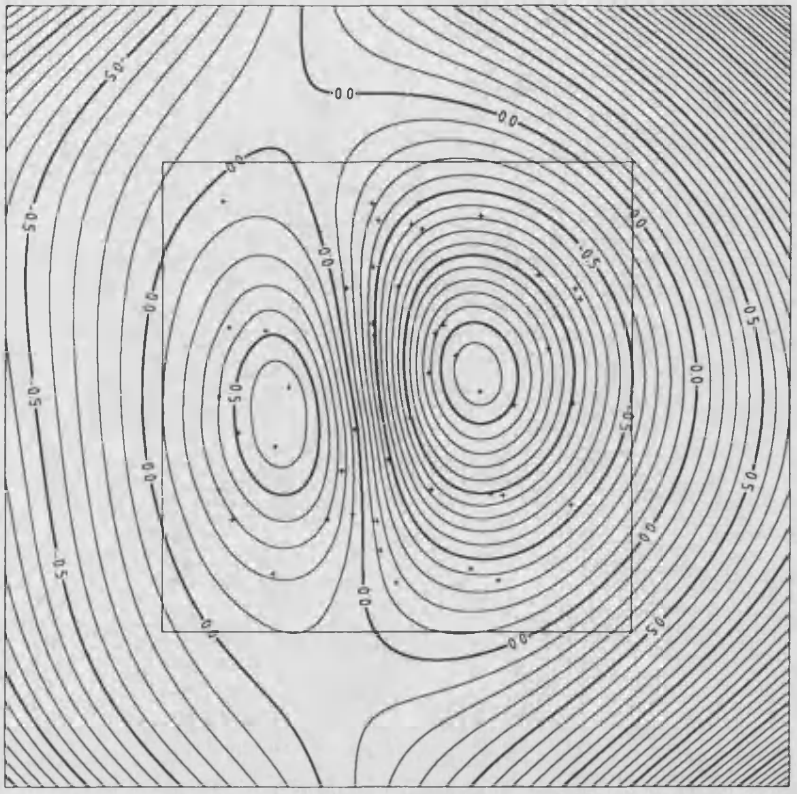


fig 3.2.1.13. 50 points, 15 extra functions, region 2

boundary which was partly finite and partly infinite (e.g. a half plane), but this would lead to some mixed boundary conditions in the differential characterisation. The solution, even approximately, of this characterisation and the practical application of the resultant splines are beyond the scope of this thesis. For our purposes we assume that Ω is either $\mathbf{R}^2$ or closed and bounded. In fact, as before we restrict ourselves to either $\Omega \equiv \mathbf{R}^2$ or Ω having a piecewise linear boundary.

The simulations carried out consist of 25 and 50 points scattered in a U-shaped sub-region of the square $\{-1.5 \leq x,y \leq 1.5\}$. Data is taken from two functions used in Sizetest namely f_1 and f_2. Splines are then fitted over the U-shape and the enclosing square using 7 and 15 extra functions, and roughnesses and recovery errors then measured. These are given in tables 3.2.2.1. and 3.2.2.2.

Looking at the roughness tables we again see that by far the largest part of the reduction in roughness occurs when going from $\Omega \equiv \mathbf{R}^2$ to either of the bounded Ω, and the differences between the bounded Ω are small compared with this reduction of between 20% and 35%. This result is supported by the recovery error figures which again show big reductions when a finite window is used and little difference between finite windows.

One interesting point is that in all cases the 7 extra function splines do better for recovery error than the 15 extra function splines. This seems to suggest that the 15 extra function splines are slightly over-smoothing in this case. However the differences are slight when compared with the differences between thin plate and finite window splines.

Figure 3.2.2.1 shows the 25 point thin plate interpolant to f_1 and figure 3.2.2.2 the 25 point interpolant with Ω the square and 7 extra functions. Figure 3.2.2.3 shows the 25 point interpolant over the U-shape shown using 7 extra functions. The contour maps for f_2 show similar, less pronounced, effects.

3.2.3. Conclusions of Sizetest and Shapetest.

The studies Sizetest and Shapetest have something to say about which region it is best to fit finite window splines over, and which functions we might expect finite window splines to recover best. Shapetest indicates that the shape of the region is largely unimportant, so that there is no need for the use of complicated regions.

Sizetest indicates that the size of the region is mostly unimportant provided that the region is reasonably close to the data. However, the investigation also indicates that a region too close to the data can cause the method to be more sensitive to the choice of number of extra functions. This second point is

Table 3.2.2.1.

Roughnesses of Splines Fitted in Shapetest.

No. of Points.	Spline over Region.	Function f_1 Roughness calculated over		Function f_2	
		U-shape	Square	U-shape	Square
25	R^2	276.3	340.3	6882	7669
	U-shape (7)	179.9	243.2	5008	6020
	(15)	172.7	239.8	4610	5814
	Square (7)	177.0	239.2	4917	5879
	(15)	176.0	238.0	4706	5715
50	R^2	333.9	392.1	9342	10153
	U-shape (7)	208.3	271.3	6631	7738
	(15)	206.6	271.6	6310	7473
	Square (7)	208.8	270.9	6574	7649
	(15)	210.1	269.9	6422	7412

Table 3.2.2.2.

Value Recovery per Unit Area in Shapetest.

No. of Points.	Spline over Region.	Function f_1 Error calculated over		Function f_2	
		U-shape	Square	U-shape	Square
25	R^2	0.599	0.470	7.79	6.43
	U-shape (7)	0.070	0.055	2.32	2.13
	(15)	0.110	0.087	2.78	2.39
	Square (7)	0.093	0.073	2.64	2.35
	(15)	0.111	0.087	3.02	2.56
50	R^2	0.106	0.084	2.87	2.79
	U-shape (7)	0.009	0.007	0.53	0.57
	(15)	0.012	0.010	0.74	0.67
	Square (7)	0.011	0.008	0.58	0.60
	(15)	0.013	0.010	0.85	0.78

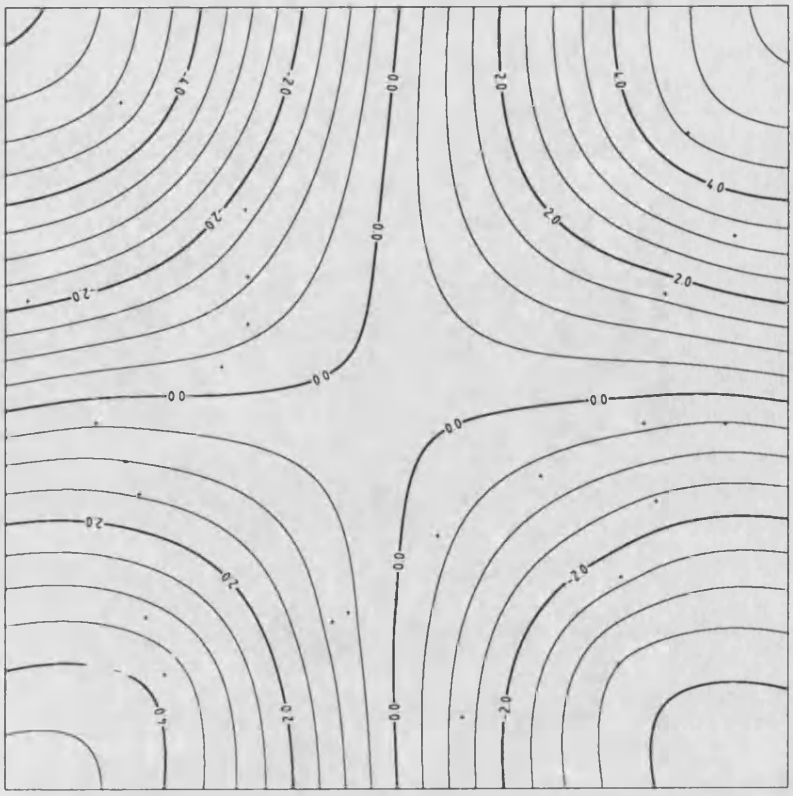


fig 3.2.2.1. 25 points, 0 extra functions

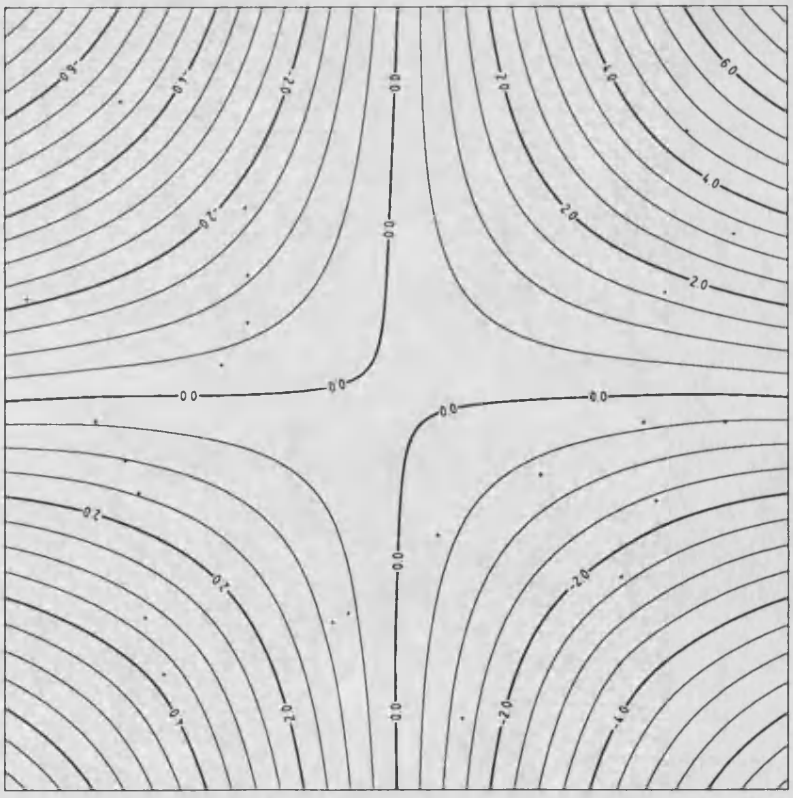


fig 3.2.2.2. 25 points, 7 extra functions, square region

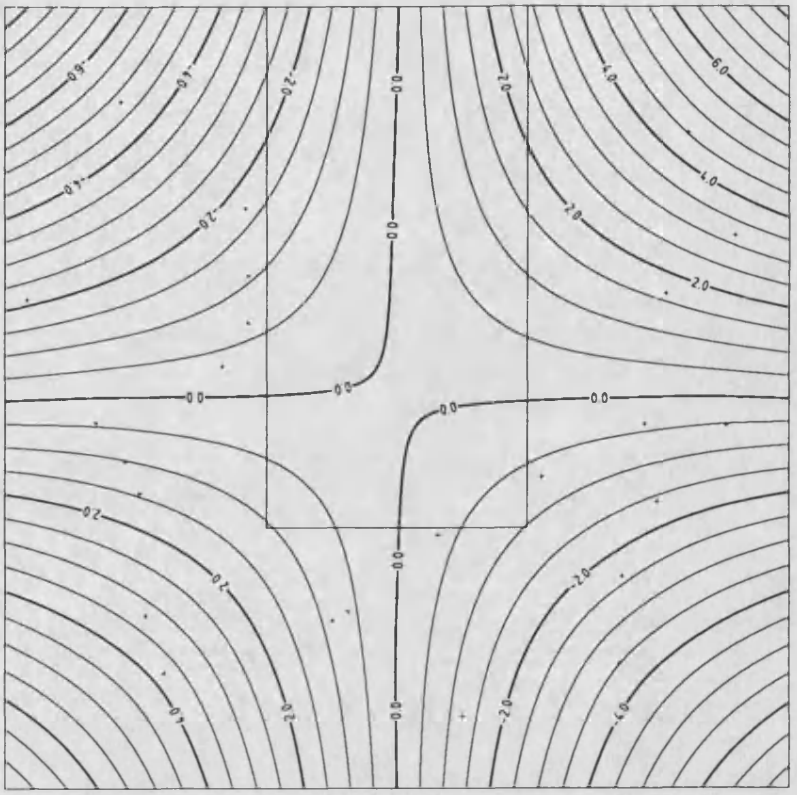


fig 3.2.2.3. 25 points, 7 extra functions, U-shaped region

supported by the fact that points close to the boundary, have ψ (that is $r^2\log r^2$) functions involved in integrals along the boundary, causing the integrands to have wide variation near the data point in question. This means that these integrals are difficult to calculate numerically, especially using simple integration rules. This problem is reduced by our techniques for evaluating the integrals, but these integrals will still have proportionately larger numerical errors, when compared to the errors in more well behaved integrals, involving only data points away from the boundary.

Regions with boundaries far from the data begin to have associated splines which return to the behaviour of thin plate splines, so an ideal region is one relatively close to the data, but not too close! We recommend a boundary which is on average about half the distance from the data, as the data is from itself. This then means that if the data were reflected in the boundary, and the boundary then removed the spread of the extended data would appear to be as dense as the original data. In this case the boundary allows about the right amount of space around the data.

It is interesting to note the type of function that we might expect finite window splines to recover best. The discussions earlier about functions satisfying the boundary conditions of finite window splines being recovered best, are of course obvious. It is certainly to be expected that a finite window spline will recover something approximating a finite window spline, better than a thin plate spline. Apart from this the simulations seem to indicate that the finite window splines will do better where the roughness of the function to be recovered is evenly spread over the region of interest. In the next section we will investigate this aspect further.

3.3. The Application of Thin plate and Finite Window Splines to a Real Data Set.

The functions used in the simulations in the previous sections, cannot be regarded as typical of the surfaces encountered in common applications of surface fitting. For example in geological applications, reconstruction of the surface of a buried rock layer using bore-hole information, is unlikely to produce a solution with as simple a shape as $4xy$. Nor is it likely to produce a solution with such locally restricted variation as the function f_5 in *Sizetest*. The function $4xy$ is a single saddle point centered at the origin, whereas f_5 consists of a steep hill and a deep pit very close to each other in an otherwise flat landscape. Real data is more likely to produce a surface which is a combination of saddles, hills and pits throughout the whole region of interest.

To compare the results of the thin plate and finite window splines, in these circumstances we have used the data taken from O'Connor and Leach (1979). This data arises from the analysis of 38 samples from a mine in Cobar, NSW, Australia. At each sampling point the 'true width' of an ore bearing rock layer and three types of ore content were measured. So in this case we have four data sets where the actual data sites are common between them. These data sets are listed in table 3.3.1.

Looking at the data we notice that it is all rounded to 2 or 3 significant figures and so, there is likely to be at least some measurement error in the data. Despite this, so as to avoid the problem of choosing a suitable smoothing parameter for the time being, we will first look at interpolating splines. Figure 3.3.1 shows the positions of the data sites and a convex region around them chosen so as to lie about the same distance from the data as half the distance between the data sites. Notice how this method of choosing the region to suit the data leads to a more even coverage than the method used in the simulations, of scattering the data 'uniformly' over a predetermined region. The fact that the positions of the data sites are fixed by a human investigator also leads to a more 'uniform' spread of the data than when the sites are chosen at random 'uniformly'.

Figure 3.3.2 shows the interpolant to the 'true width' (data set 1) using $\Omega \equiv \mathbf{R}^2$. The effects of rounding can be seen in that a large number of the data sites lie exactly on contours. When approximating the finite window spline we have to decide how many extra functions to use. In the simulated data this was done by comparing the splines with 7 and 15 extra functions, since we suspect the real data case to be more complicated, we must be more careful about this choice. There are two important criteria which may be used to determine a choice for n, the number of extra functions. These are the convergence of the reductions in roughness and the convergence of the solution spline itself. The second of these criteria is more difficult to examine since it depends upon the behaviour of a whole surface as opposed to a single real number in the roughness criterion. Also since it is this residual roughness the method is designed to minimize we chose the roughness criterion as our means of selecting n.

Table 3.3.2 shows the residual roughness of the interpolants for each data set and several values of the parameter n. From the table it appears that in all but data set 1 the roughnesses have started to converge at 31-39 extra functions. However data set 1 has an erratic behaviour upto 27 extra functions and decreases steadily by about 0.15 thereafter. We notice from this that even 15 extra functions may be too few to take, in the case of real data, and larger values of n are required. Also notice that the overall reductions in roughness are much smaller, than in some of

Table 3.3.1.

Positions of Data Sites and Corresponding Data Values.

	Data Site		Data Values			
No.	x	y	Set 1	Set 2	Set 3	Set 4
1	-16.0	-15.0	17.0	0.984	0.606	3.297
2	-14.0	-4.0	18.0	0.850	0.564	3.177
3	-13.0	4.0	17.5	0.957	0.259	1.691
4	-7.0	5.0	19.0	1.260	0.870	6.829
5	-6.0	-43.0	22.0	1.709	1.486	7.563
6	-6.0	-36.0	24.0	0.900	1.298	10.670
7	1.0	-50.0	17.4	0.952	1.025	12.140
8	2.0	-39.0	23.0	0.982	0.720	7.727
9	2.0	-8.0	23.5	1.773	1.464	7.329
10	2.0	-51.0	15.0	0.829	0.877	11.420
11	9.0	-16.0	23.5	1.723	2.213	9.103
12	9.0	-42.0	25.0	1.422	1.922	7.405
13	17.0	-37.0	16.5	1.264	1.551	6.038
14	18.0	-12.0	19.5	1.580	2.228	6.232
15	24.0	-57.0	12.0	1.511	0.517	3.684
16	25.0	-29.0	18.5	1.367	1.077	3.257
17	26.0	-40.0	18.0	1.757	0.409	2.809
18	32.0	-7.0	14.0	0.602	2.184	4.258
19	33.0	-35.0	19.0	1.261	0.938	5.316
20	40.0	4.0	13.5	1.859	0.642	1.281
21	40.0	-61.0	18.0	1.401	0.079	0.190
22	44.0	-29.0	19.4	1.565	0.519	1.773
23	48.0	-65.0	13.0	0.798	0.056	0.272
24	48.0	-7.0	14.0	0.971	0.860	1.727
25	49.0	-32.0	19.5	1.660	1.662	1.952
26	55.0	-71.0	16.0	1.654	0.379	0.375
27	56.0	-14.0	16.0	1.014	1.521	3.133
28	59.0	-38.0	19.0	1.171	1.510	2.840
29	62.0	7.0	19.0	0.835	0.099	3.703
30	62.0	-3.0	21.5	2.477	0.082	1.417
31	64.0	-29.0	22.0	1.917	1.818	2.601
32	69.0	-28.0	20.5	1.403	0.913	2.131
33	70.0	-72.0	11.0	1.089	0.403	3.141
34	77.0	-19.0	26.0	1.117	0.637	2.474
35	78.0	-53.0	22.0	0.628	2.331	5.186
36	79.0	-37.0	26.0	0.877	1.337	5.836
37	84.0	-52.0	16.0	0.642	1.592	11.130
38	84.0	-16.0	16.0	0.230	0.129	0.968

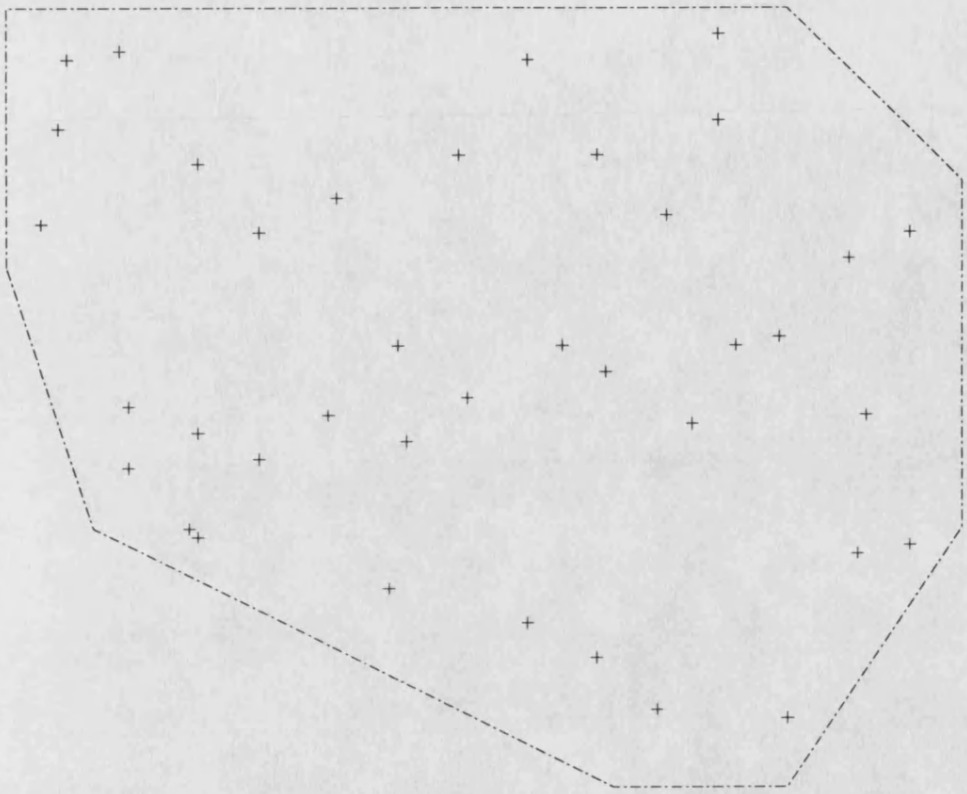


fig 3.3.1 The data sites

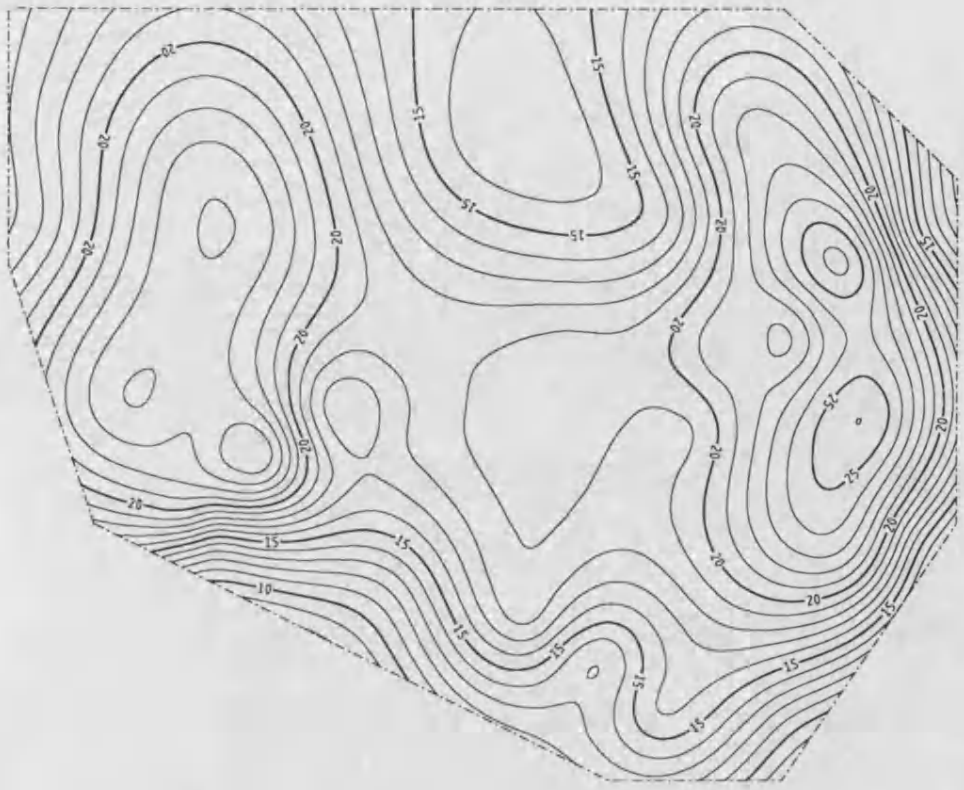


fig 3.3.2 Thin-plate interpolant (data set 1)

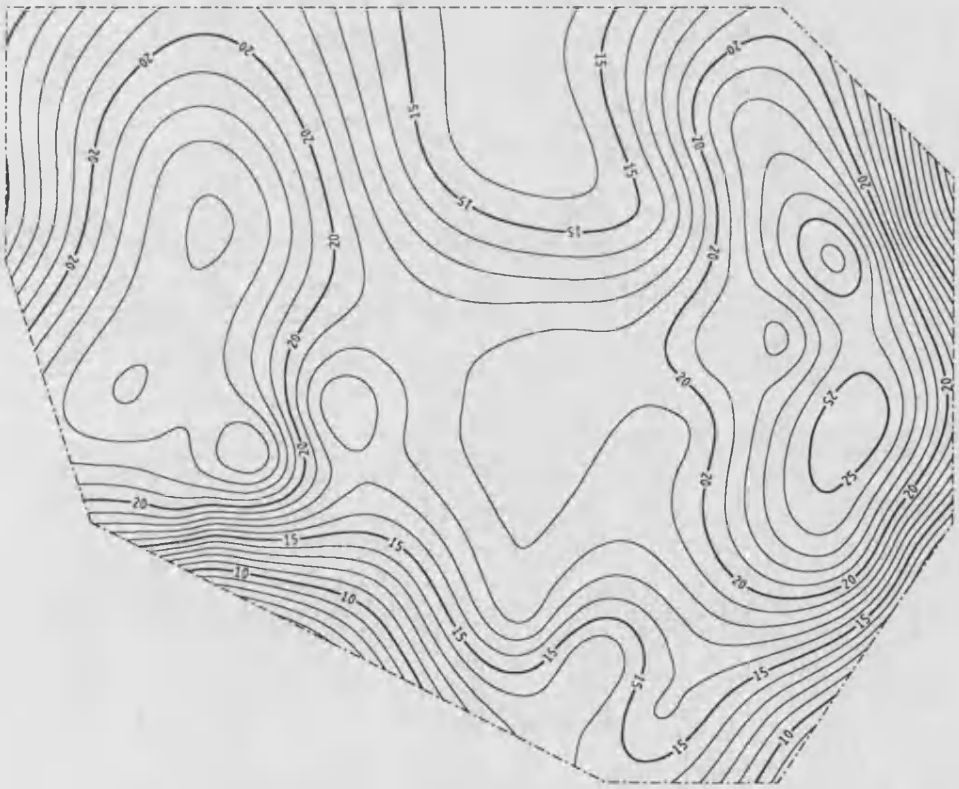


fig 3.3.3 23 extra functions interpolant

the simulations. The final number of extra functions chosen was 23 and this represents only 3%, 4.25%, 1.5% and 6.25% reductions in roughness for each data set respectively. This can probably be attributed to the fact that not only are the recovered surfaces more complicated than any of those in the simulated data, but also the more even coverage of the data sites, together with the interpolation conditions allow much less freedom in the surface to be determined by smoothness criteria. At least one of these problems should be reduced significantly when some smoothing is applied, thereby relaxing the interpolation conditions.

Table 3.3.2.
Residual Roughness of Interpolants.

No. of Extra Functions		Data Set 1	2	3	4
0	($\Omega \equiv R^2$)	64.06	1.195	1.660	21.34
3		63.81	1.192	1.657	21.32
7		63.71	1.187	1.651	20.70
11		63.45	1.176	1.646	20.63
15		62.68	1.172	1.641	20.44
19		62.61	1.153	1.640	20.11
23		62.08	1.144	1.633	20.00
27		61.98	1.140	1.631	19.95
31		61.79	1.129	1.629	19.85
35		61.66	1.128	1.625	19.80
39		61.55	1.123	1.624	19.77

Figure 3.3.3 shows the interpolant, to the first data set, using 23 extra functions and the region shown. Here, in the real data application, the differences appear to be much more confined to the edges than in the simulated data. It is difficult to see these differences, beyond the largest, in contour maps of this kind and more difficult for us to describe them. However since both splines can be described using the coefficient vectors λ, b, and d for b a 23 vector of coefficients of the extra functions (all held zero for the thin plate spline, figure 3.3.2) we can easily look at the differences between the two splines by subtracting the respective coefficients and contouring the residual function.

Figure 3.3.4 shows the contour map of the difference between the thin plate and the finite window splines (23 extra functions). A problem with these maps is that large areas are relatively flat and close to zero. This is known to be a difficult surface for contouring algorithms to map well and results in CONICON3 choosing large wiggly zero contours. The position of these zero contours is in theory variable throughout the region where the residual is zero, but in practice is

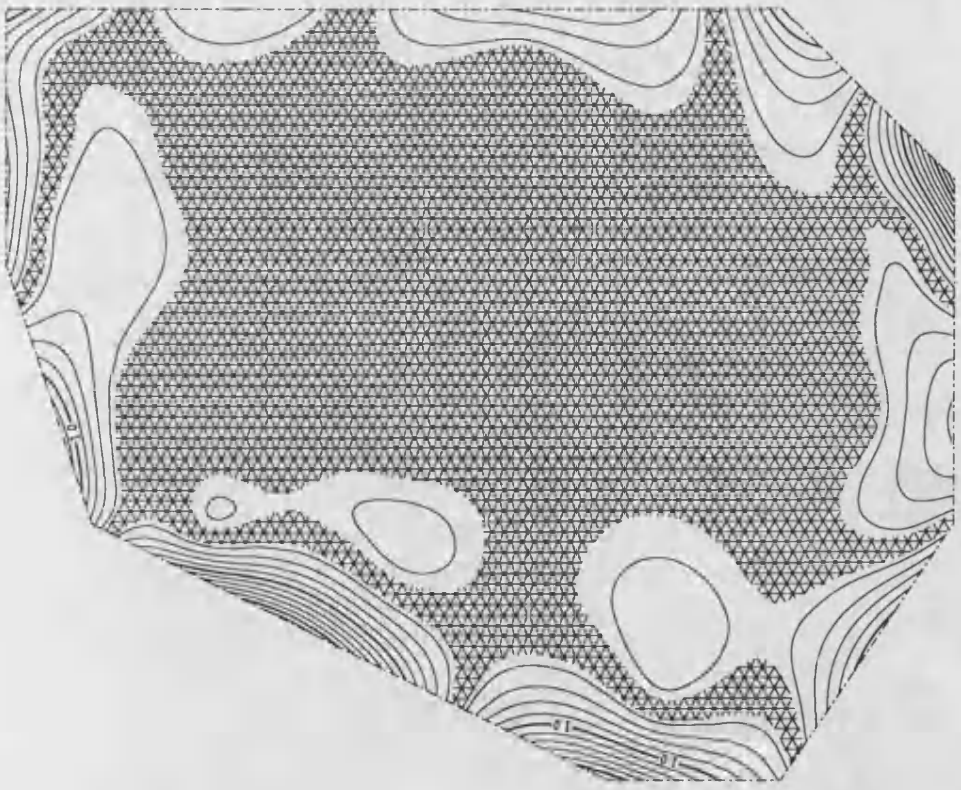


fig 3.3.4 Difference between 0 and 23 extra functions

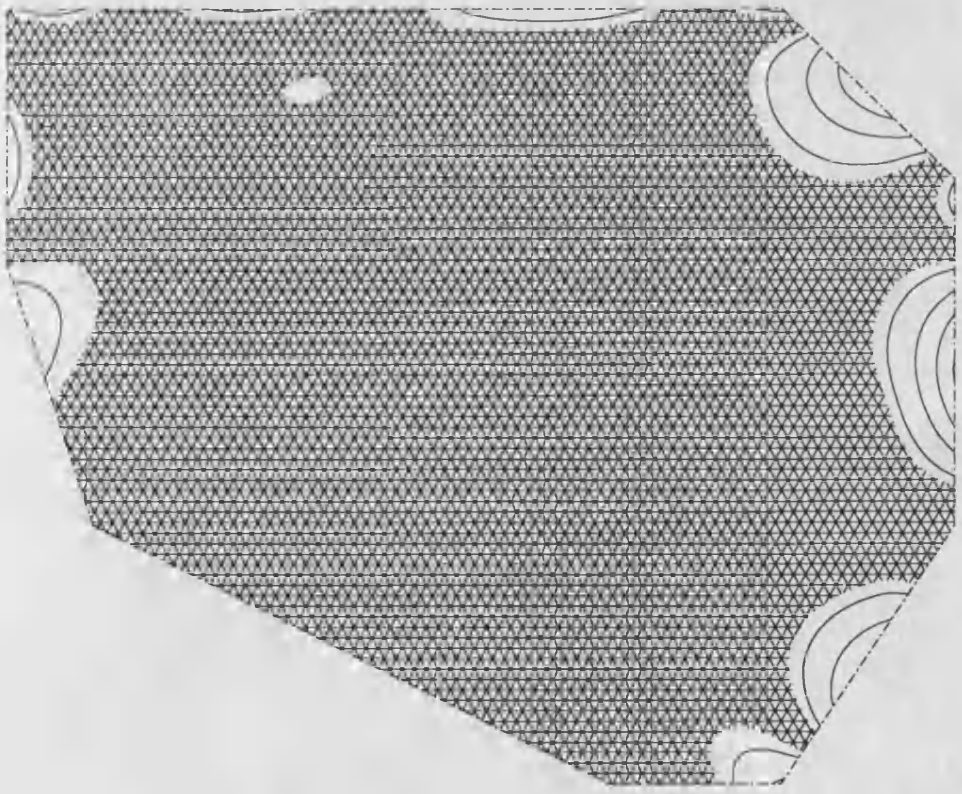


fig 3.3.5 Difference between 23 and 31 extra functions

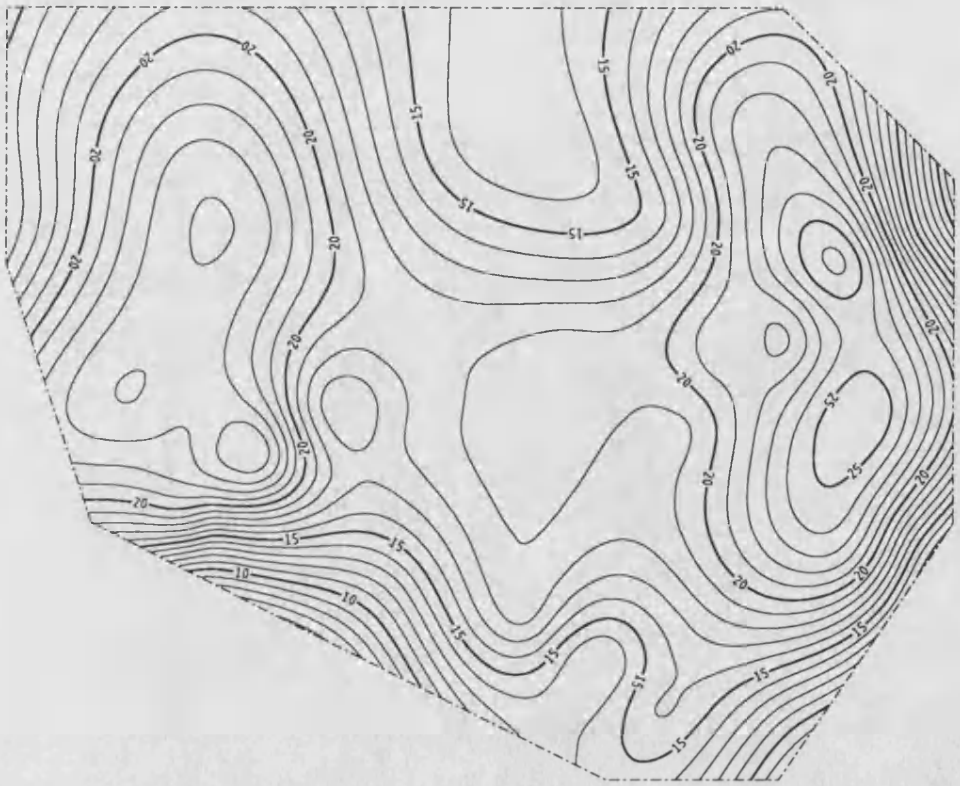


fig 3.3.6 31 extra functions interpolant

determined by the rounding errors in the splines. At the positions of the data sites the difference function is defined to be zero so almost all the sites lie under zero contours. To overcome this problem of indeterminate zero contours, the maps have been cross-hatched in regions where the function is close to zero and the zero contours omitted. (By close to zero here we mean that the absolute value of the residual function is less than about $\frac{1}{2}\%$ of the maximum value of the original splines.)

Figure 3.3.4 shows that the finite window spline procedure is indeed an edge correction procedure, although there some areas in the lower part of the data that change slightly, away from the edge.

In order to see how our roughness criterion for choosing the number of extra functions, effects the shape of the finite window spline we have contoured in the same way as above, the difference between a finite window spline with 23 extra functions and one with 31 extra functions. Again the differences can be seen, in figure 3.3.5, to be restricted to the edges of the region, even more so than in figure 3.3.4 and the differences are in general smaller. There is a problem here in that small differences in the value of the surface can result in large changes in the paths of the contours, if the gradient of the surface is small. This is apparent when comparing figure 3.3.6, the 31 extra function spline, with figure 3.3.3. At the centre-top of the region the contours appear quite different, but figure 3.3.5 shows that these differences are about a third of those between figures 3.3.2 and 3.3.3.

The other data sets show similar results to this one. The figures are presented here for completeness. The data sets 2 and 3 appear more complicated than data set 1, in that there is more variation in the region of the data. Figures 3.3.7 and 3.3.8 show the thin plate and 23 extra function splines respectively for data set 2. Immediately we see that this surface is more complicated in the central region than the first data set, and there are several differences between the two splines. Figures 3.3.9 and 3.3.10 are the differences between the thin plate and 23 extra function splines and the 23 and 31 extra function splines respectively, with the shaded region determined as before. It can be seen that, although the splines have not really converged yet, there are extensive areas around the edges of the region and away from the data sites where the thin plate and finite window splines differ.

Figures 3.3.11 to 3.3.14, for data set 3, show much the same effects as those for data set 2. Data set 4 shows a less erratic variation than any of the previous data sets, and the thin plate and 23 extra function splines are plotted in figures 3.3.15 and 3.3.16. Figure 3.3.17 shows probably the largest area of differences between

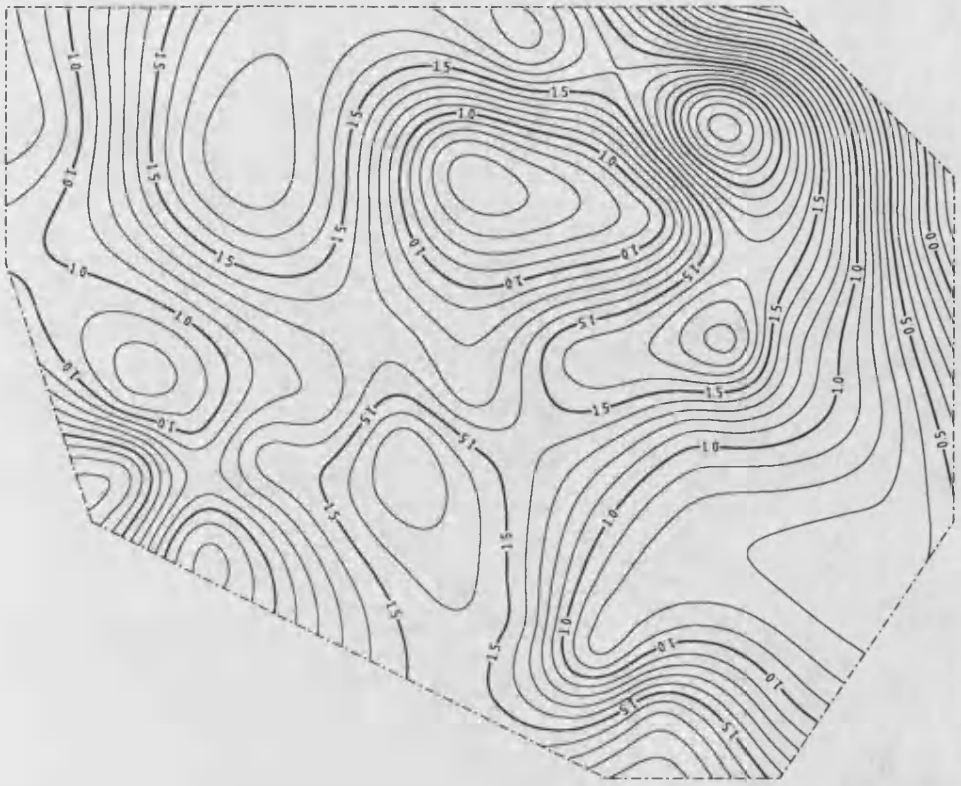


fig 3.3.7 Thin-plate interpolant (data set 2)

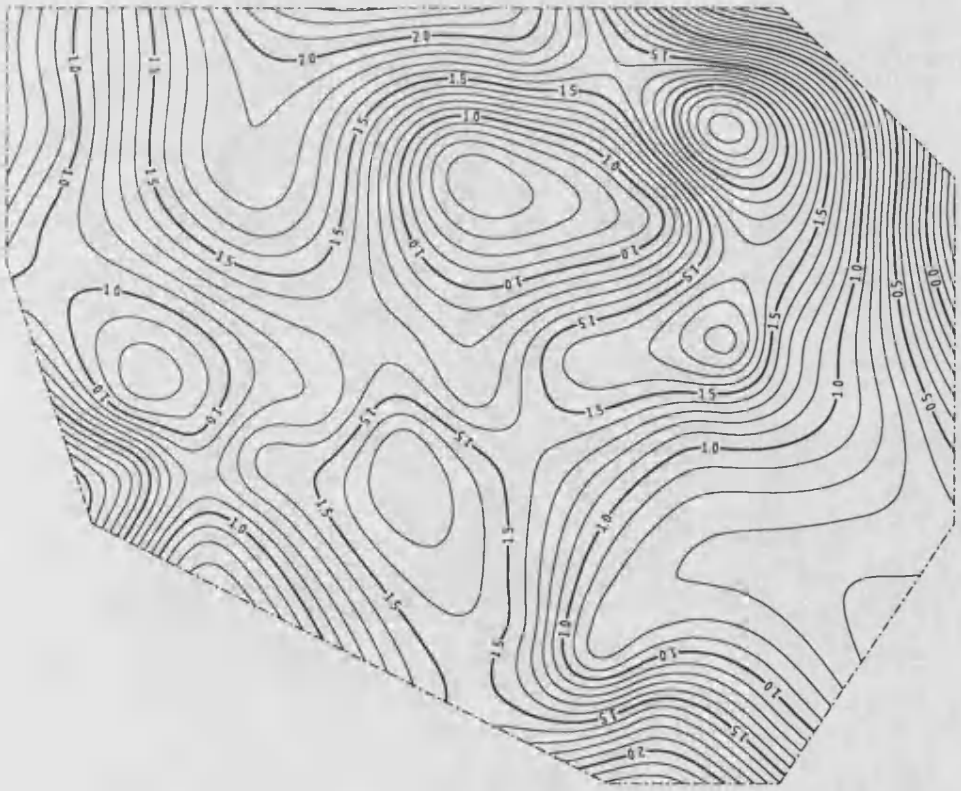


fig 3.3.8 23 extra functions interpolant

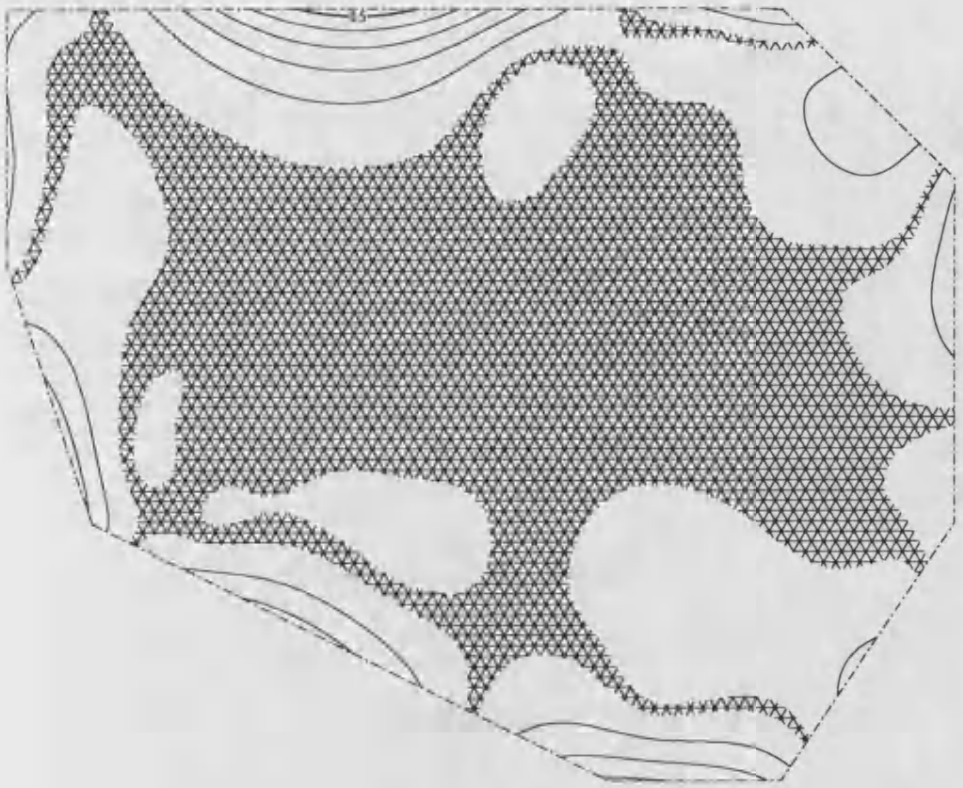


fig 3.3.9 Difference between 0 and 23 extra functions

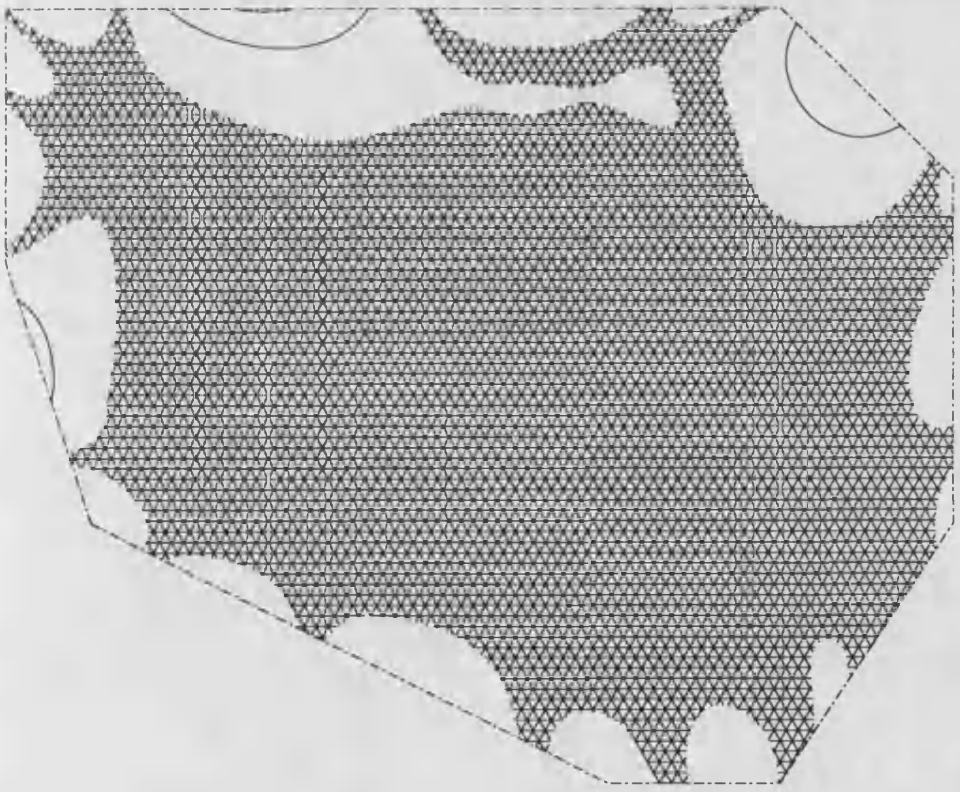


fig 3.3.10 Difference between 23 and 31 extra functions

fig 3.3.11 Thin-plate interpolant (data set 3)

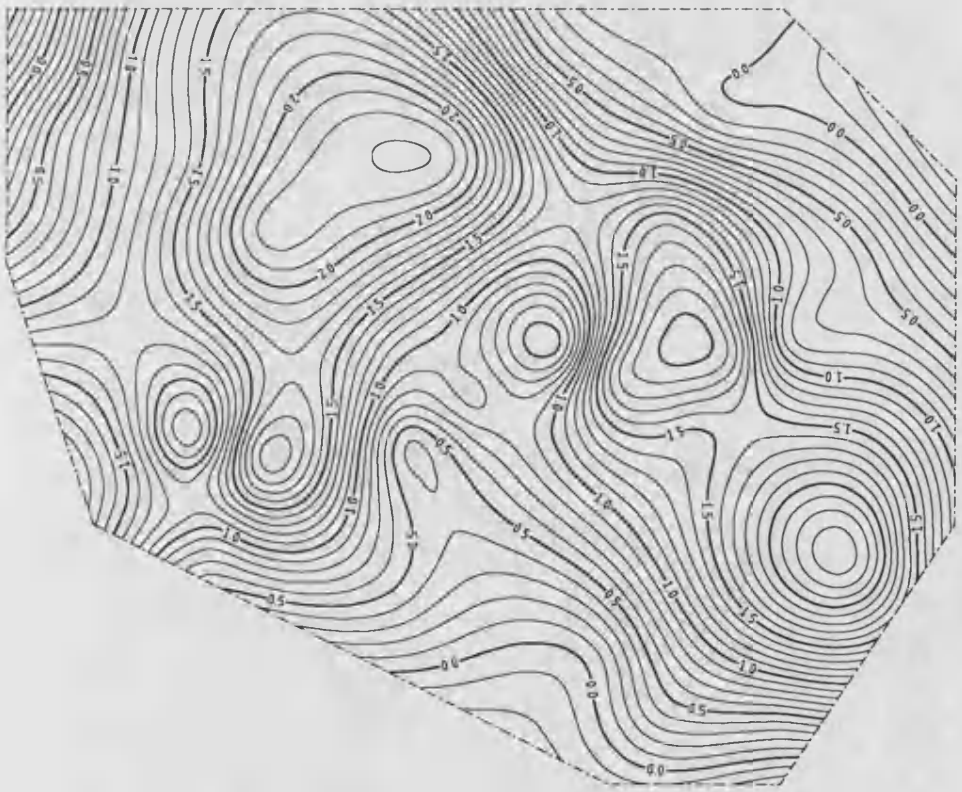


fig 3.3.12 23 extra functions interpolant

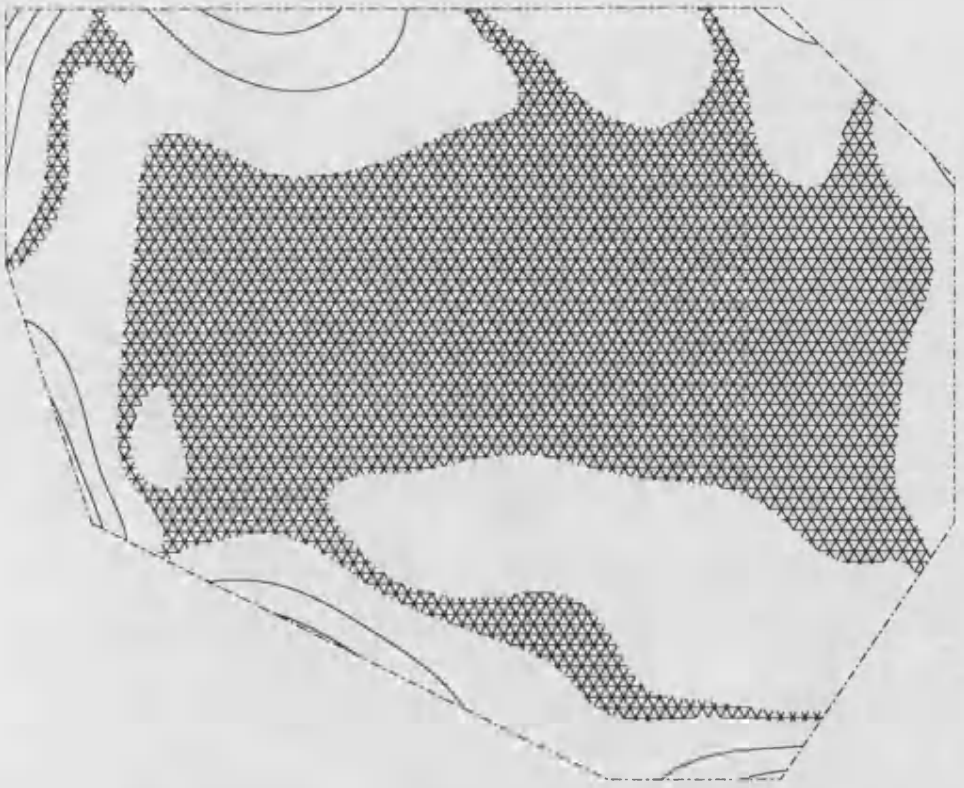


fig 3.3.13 Difference between 0 and 23 extra functions

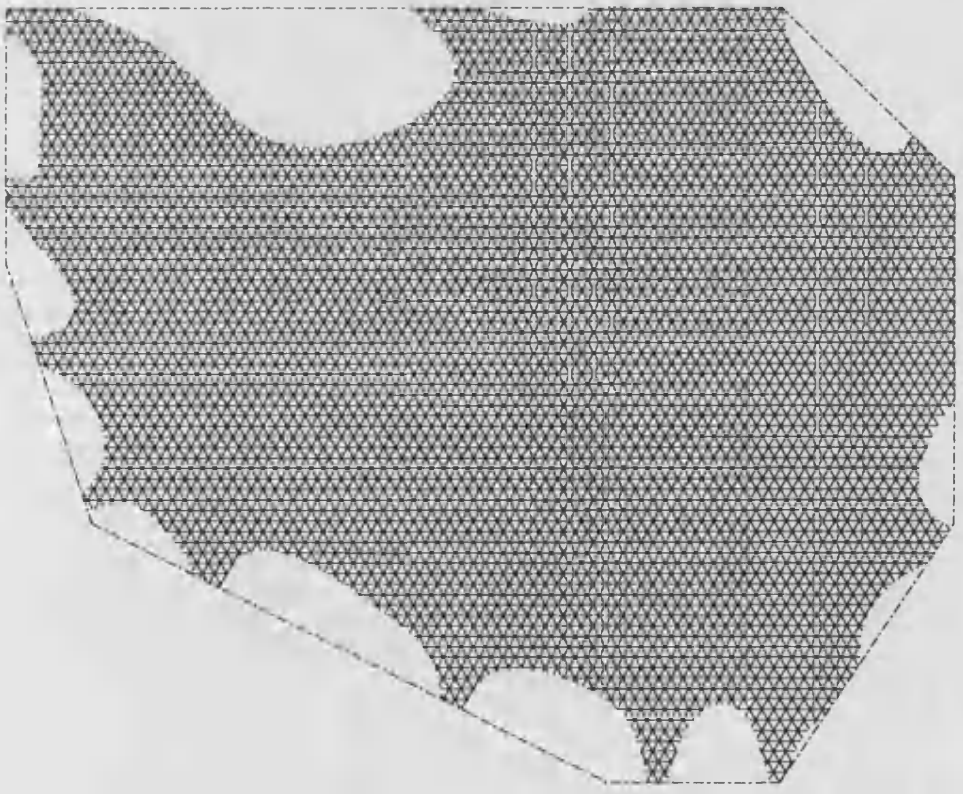


fig 3.3.14 Difference between 23 and 31 extra functions

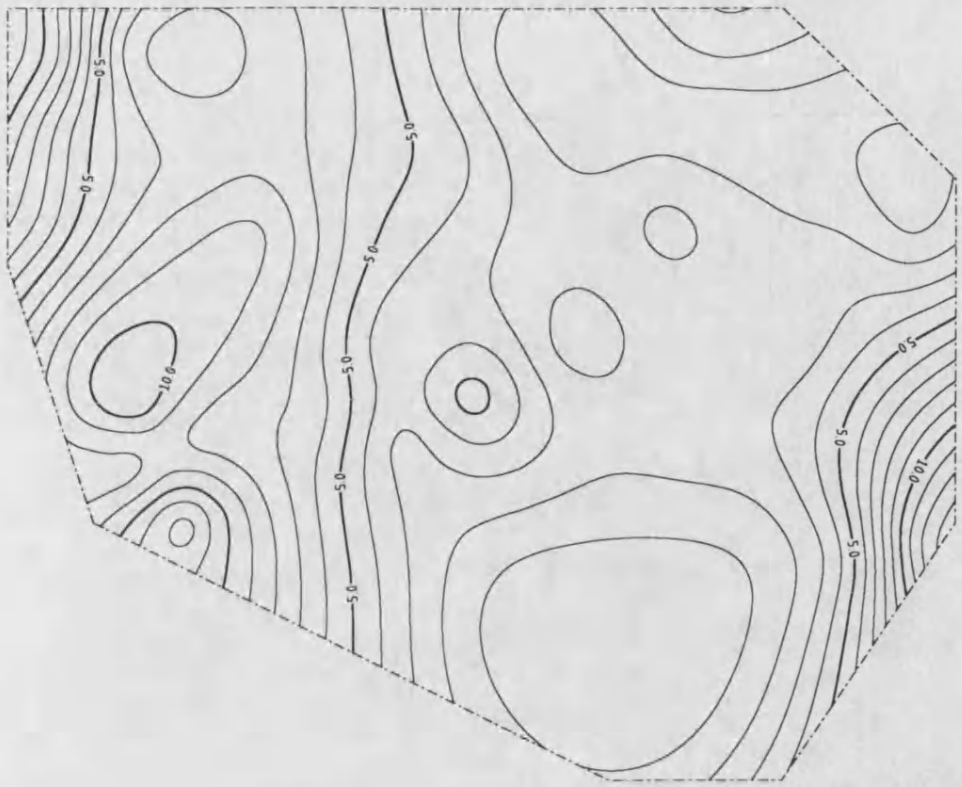


fig 3.3.15 Thin-plate interpolant (data set 4)

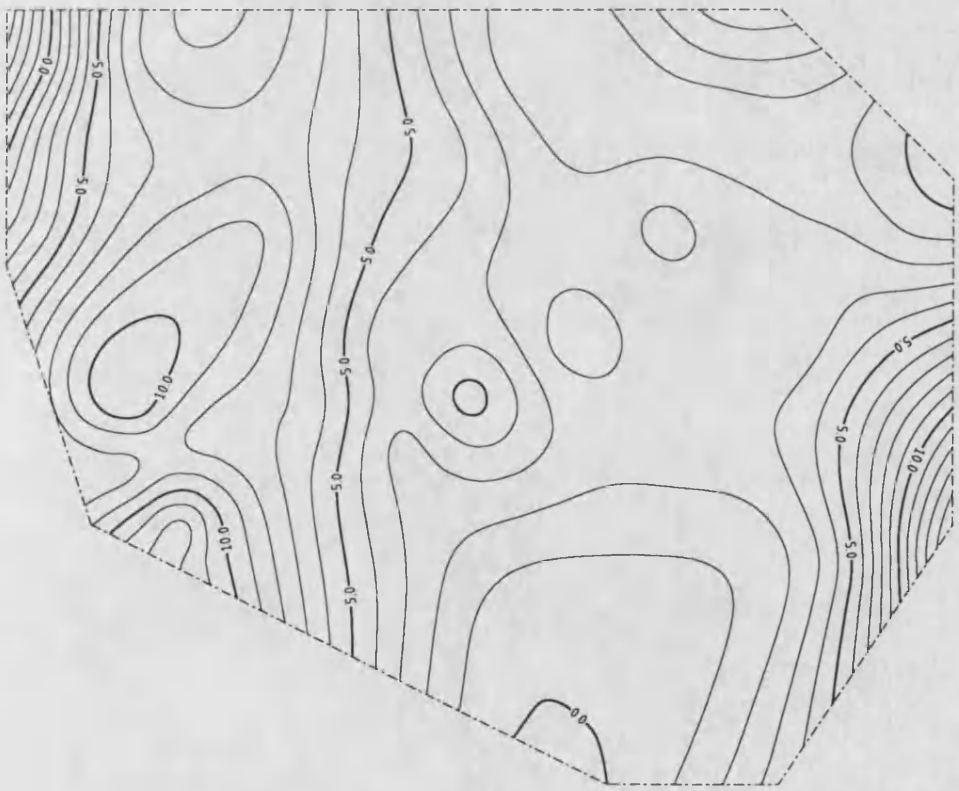


fig 3.3.16 23 extra functions interpolant

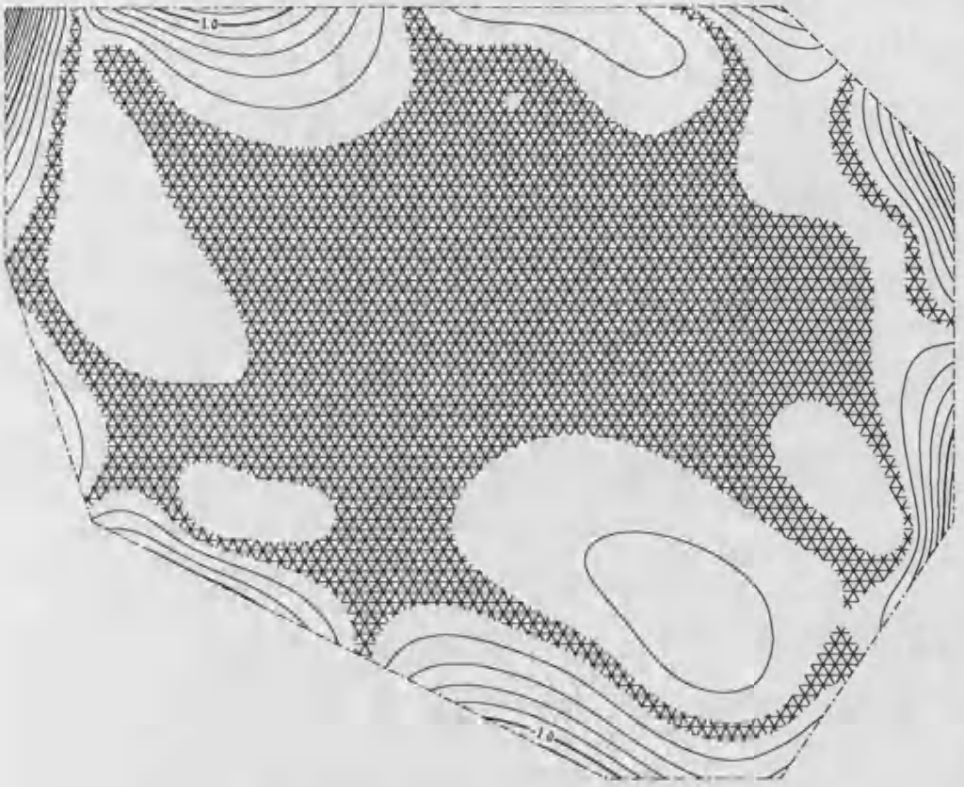


fig 3.3.17 Difference between 0 and 23 extra functions

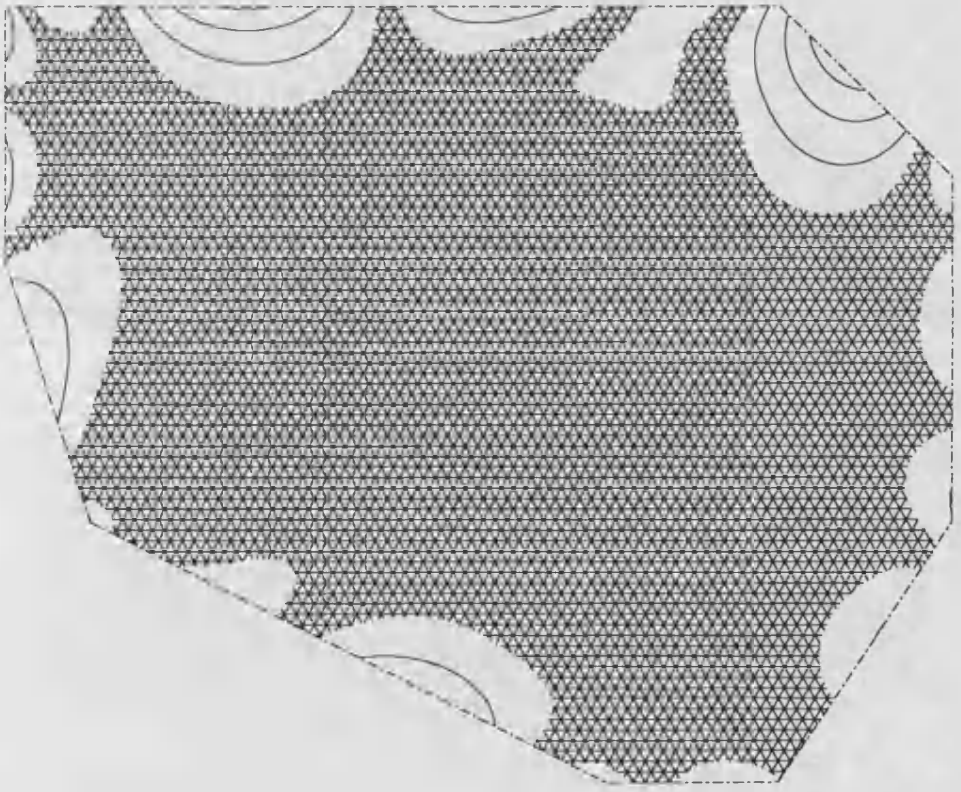


fig 3.3.18 Difference between 23 and 31 extra functions

these two splines of all the data sets, while figure 3.3.18, the difference between the 23 and 31 extra function splines, shows similar results to the equivalent previous plots.

Conclusions.

The application of finite window splines to real data can largely be regarded as an edge correction procedure. Despite the difficulty of obtaining convergence to the *true* finite window spline using the techniques of Dyn and Levin outlined here, fitting modest numbers of extra functions can produce splines significantly different in appearance from thin plate ($\Omega \equiv \mathbf{R}^2$) splines, at least away from the data, in the interpolation case. However closer to the data the differences are minimal, and this is to be expected in the interpolation case as the interpolation conditions tie down the fitted function in regions close to the data sites.

In general the effects seen in the real data examples are much less marked than the effects observed in the simulations. This might be seen as a disappointing result until it is remembered that, although interpolation is important in bivariate regression, in order to look at the data effectively, it is hardly likely to be a wholly appropriate procedure for modelling surfaces based on real data. In this case the smooth fitted surface may be unduly affected by errors in the recorded data and smoothing these data values is desirable in order to estimate the *error free* values, in all cases except where the errors are known to be small. Even in the case of small errors, smoothing may be appropriate in order to extract trend information from the data more effectively.

In the next section we shall briefly look at some smoothed thin plate and finite window splines, fitted to the real data of this section.

3.4. Finite Window Smoothing Splines.

The mathematics and implementation of finite window smoothing splines was discussed in earlier chapters. Here we look at their application to the first of the real data sets mentioned in the previous section. The automatic choice of smoothing parameter does not concern us here, since we are interested in comparing thin plate and finite window splines, where a similar amount of smoothing has been performed. There is a problem in matching the amounts of smoothing carried out by the different types of splines. We cannot simply use the same value of smoothing parameter, as this parameter biases a different measure of roughness in the two spline problems.

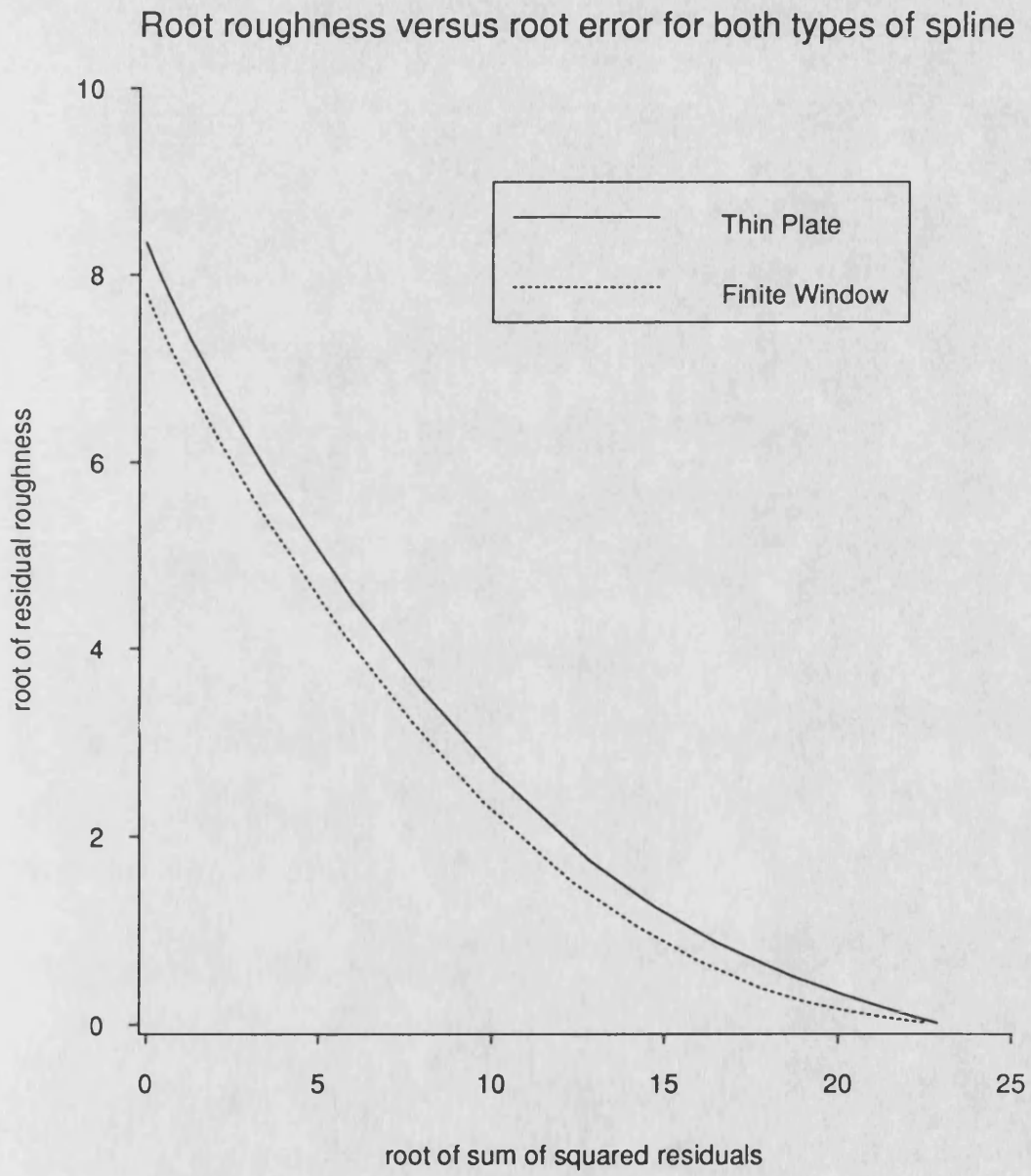
Recall that the functional being minimised can be written as,

$$E(u)+\alpha J_{\Omega}(u)$$

where E is an error penalty, usually the sum of squared residuals, and J_{Ω} is a measure of roughness, namely the integral over Ω ($\equiv \mathbf{R}^2$ or some bounded region) of the sum of all the second derivatives squared. Thus when $\Omega \equiv \mathbf{R}^2$ the smoothing parameter is biasing a larger factor than when Ω is some bounded subset of $\mathbf{R}^2$.

There are at least two ways of assessing the amount of smoothing that the splines have done to the data. The first that we shall consider is the error. With this we measure the sum of squared residuals of the fitted smoothing spline, and use this as an indicator of the amount of smoothing performed. We also consider the residual roughness over the region Ω over which the finite window spline is fitted as a measure of smoothing. This measure says that two splines have performed the same amount of smoothing if they have the same residual roughness over the region. This choice of region is preferable rather than the choosing $\mathbf{R}^2$ since, although the finite window spline is defined outside the associated region, its roughness over the whole plane its not necessarily finite.

Before proceeding to compare the two types of splines, using the above measures to determine comparable amounts of smoothing, we shall look at the interaction of these measures. For a range of values of the smoothing parameters the residual roughnesses and errors were calculated for both types of spline. Figure 3.4.1 shows the graphs of the square root of residual roughness plotted against the square root of the sum of squared residuals, for both spline types. The solid line is for thin plate splines. As the smoothing parameters increase, the errors and roughnesses change in such a way that we move, at differing rates, from left to right along the respective curves. At the far right of the curves, where the roughnesses are zero, the fitted surfaces are both planes. In fact the splines are both the same, and are the least squares fitted plane. However away from this point, the curves diverge with the finite window spline lying below the thin plate spline curve. This can be interpreted in two ways. The finite window spline has lower residual roughness for a given residual sum of squares, or the finite window spline has lower error for a given residual roughness. Also it can be seen, in this example, that the difference between the two curves is almost constant, so that proportionately the differences in roughness for a given error, are greater where more smoothing has taken place.


Root roughness versus root error for both types of spline
Thin Plate
Finite Window
root of residual roughness
root of sum of squared residuals
0
2
4
6
8
10
0
5
10
15
20
25
figure 3.4.1

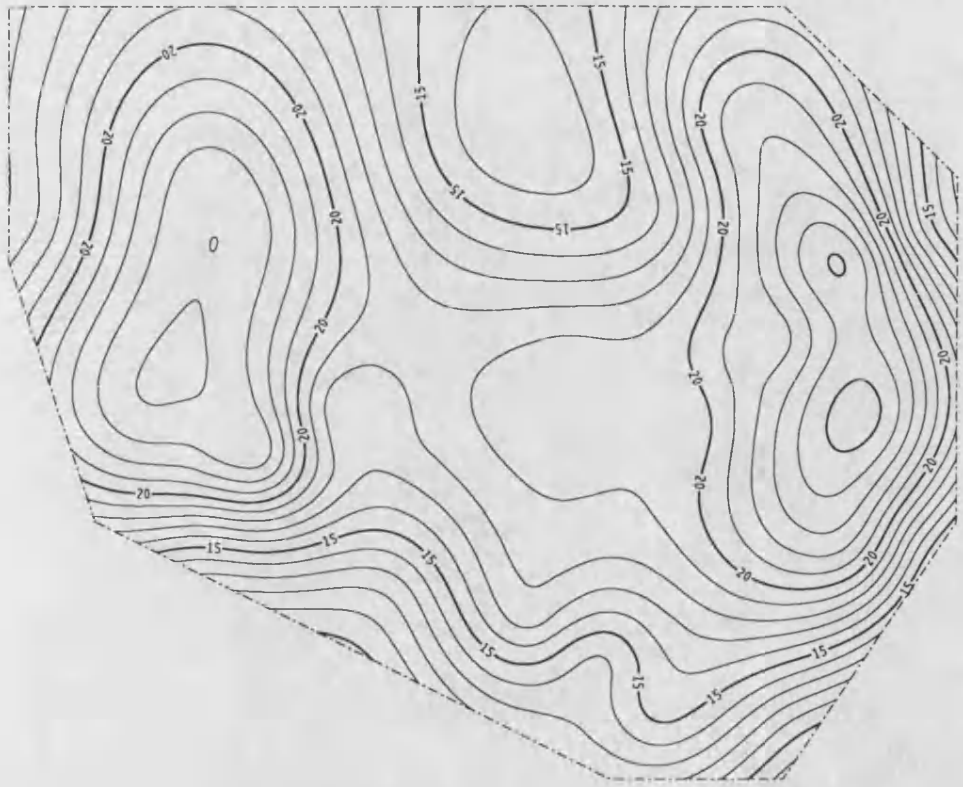


fig 3.4.2 Thin-plate smoother (alpha=1)

fig 3.4.3 Error matched (23 extra functions)

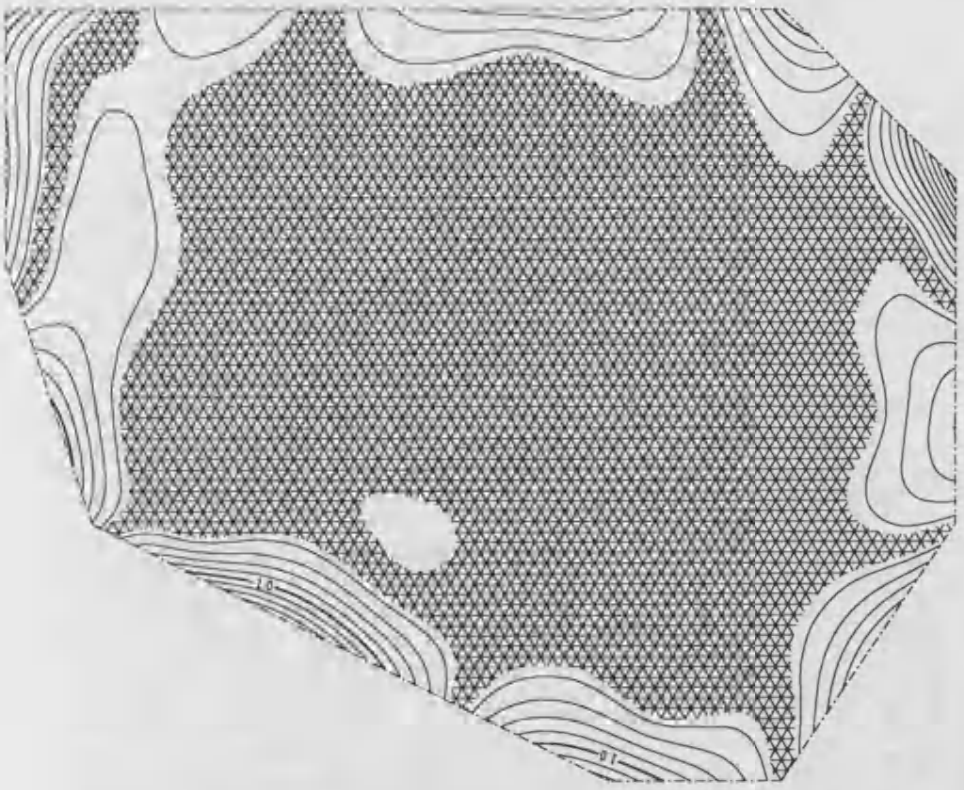


fig 3.4.4 Difference between thin-plate and error matched splines

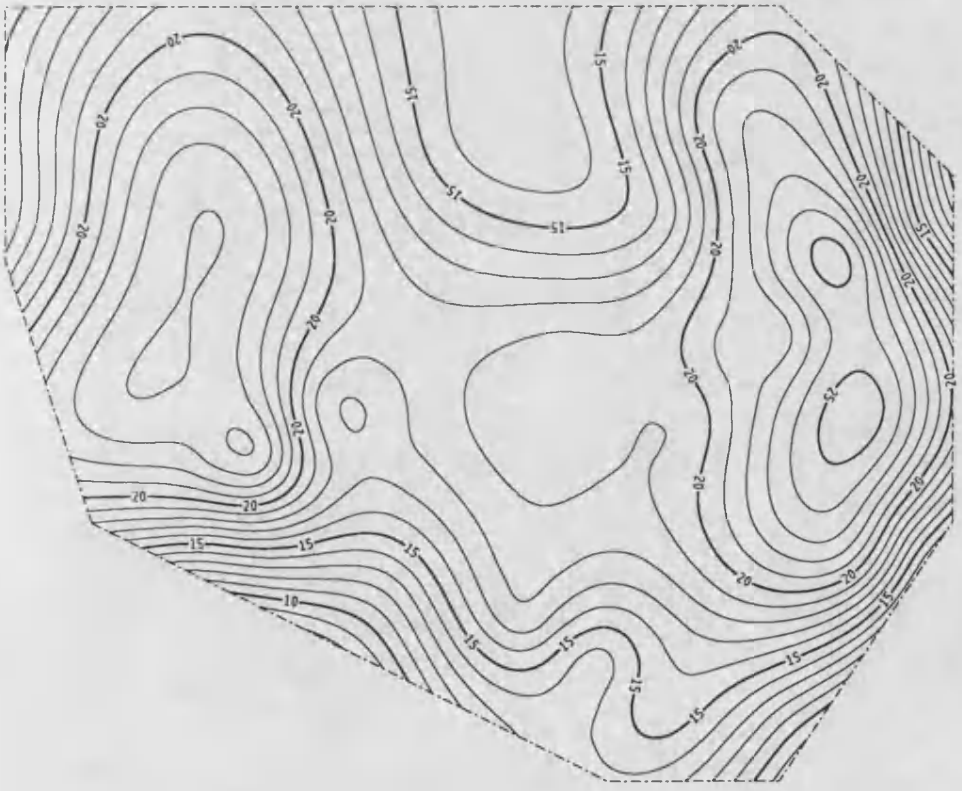


fig 3.4.5 Roughness matched (23 extra functions)

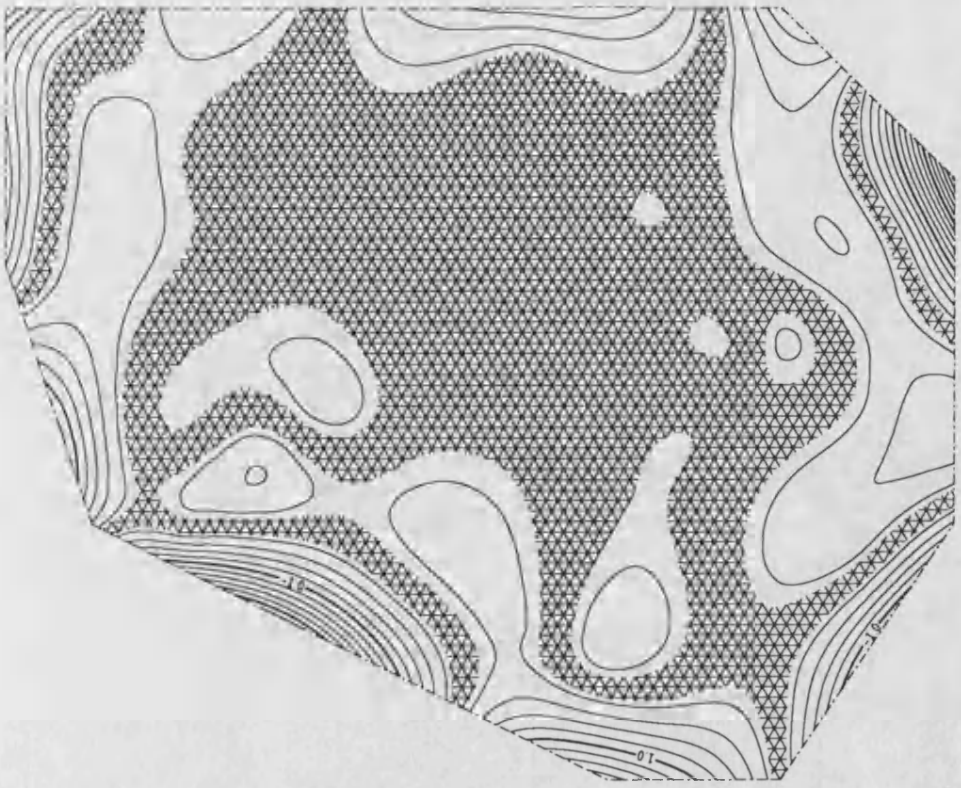


fig 3.3.6 Difference between thin-plate and roughness matched splines

To compare thin plate and finite window smoothing splines, three values of the smoothing parameter were chosen in thin plate case, namely 1, 10 and 100, and the finite window splines with the same error and the same roughness were found for each thin plate spline. Figure 3.4.2 shows the thin plate spline smoother with the value of the smoothing parameter 1. The residual roughness of this spline is 34.67 and the sum of squared residuals, is 12.24. Figure 3.4.3 shows the error penalty matched, 23 extra function finite window smoothing spline, and figure 3.4.4 shows the difference between the two splines, with the hatched area determined as before. Figures 3.4.5 and 3.4.6 show the roughness penalty matched finite window spline and the difference between this and figure 3.4.2. Figures 3.4.7 to 3.4.11 and 3.4.12 to 3.4.16 show the equivalent pictures for thin plate smoothing parameters of 10 and 100 respectively. Table 3.4.1 below shows the roughnesses and error penalties of all the splines fitted.

Table 3.4.1.

	Thin Plate		Finite Window	
α	Roughness	Error Penalty	Error Matched Roughness	Roughness Matched Error Penalty
1	34.67	12.24	27.53	6.23
10	7.10	102.24	4.41	73.97
100	0.77	272.83	0.31	215.95

The table demonstrates also, that the finite window spline has less residual roughness for a given error and less residual error penalty for a given roughness, than the thin plate spline, as figure 3.4.1 has shown.

For both the error penalty matching and roughness penalty matching splines, the differences between the thin plate and finite window versions become greater and more extensive, as the smoothing is increased, in our examples. The differences in the roughness matched splines are all much larger than the differences in the error matched splines, and the increase in the differences is more marked when comparing the roughness matched splines. This is to be expected since the error matched splines are at least matching by a function of the values of the spline at the data sites, and since the plotted differences are differences in value, we expect the error matched splines to be closer than the roughness matched splines.

One interesting observation can be made if we look back at the difference between the interpolating thin plate and finite window splines for this data set (figure 3.3.4). Comparing this with figure 3.4.4 we see that the difference between the error matched splines at the lowest level of smoothing tried, is slightly less than the difference between the interpolating splines. We also know

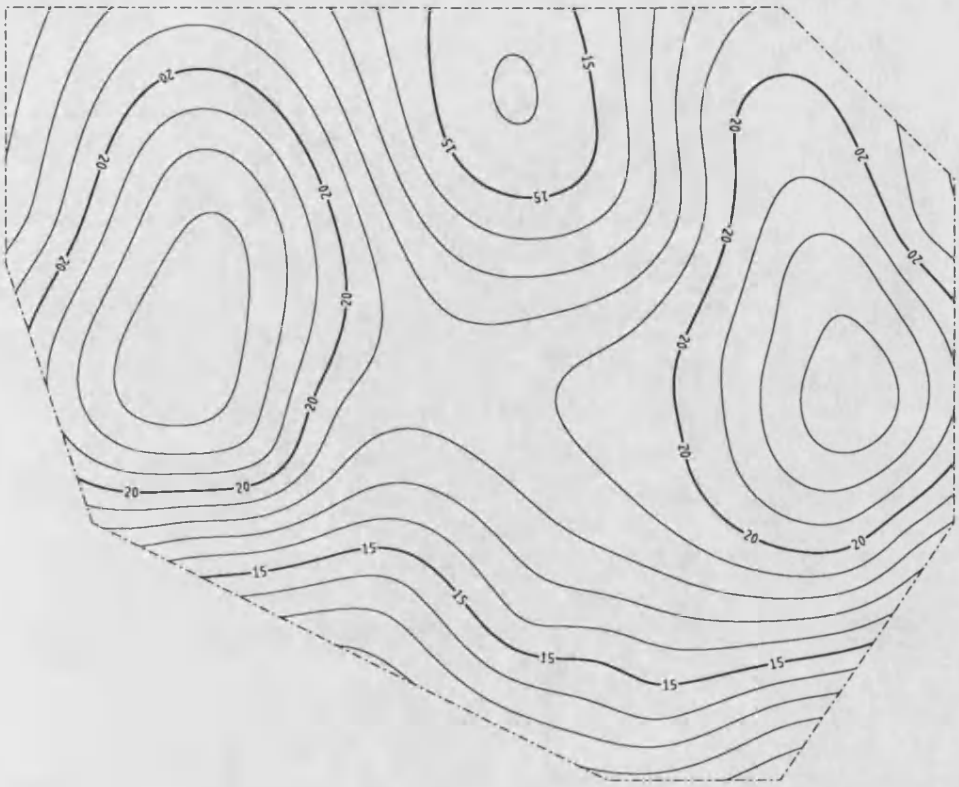


fig 3.4.7 Thin-plate smoother (alpha=10)

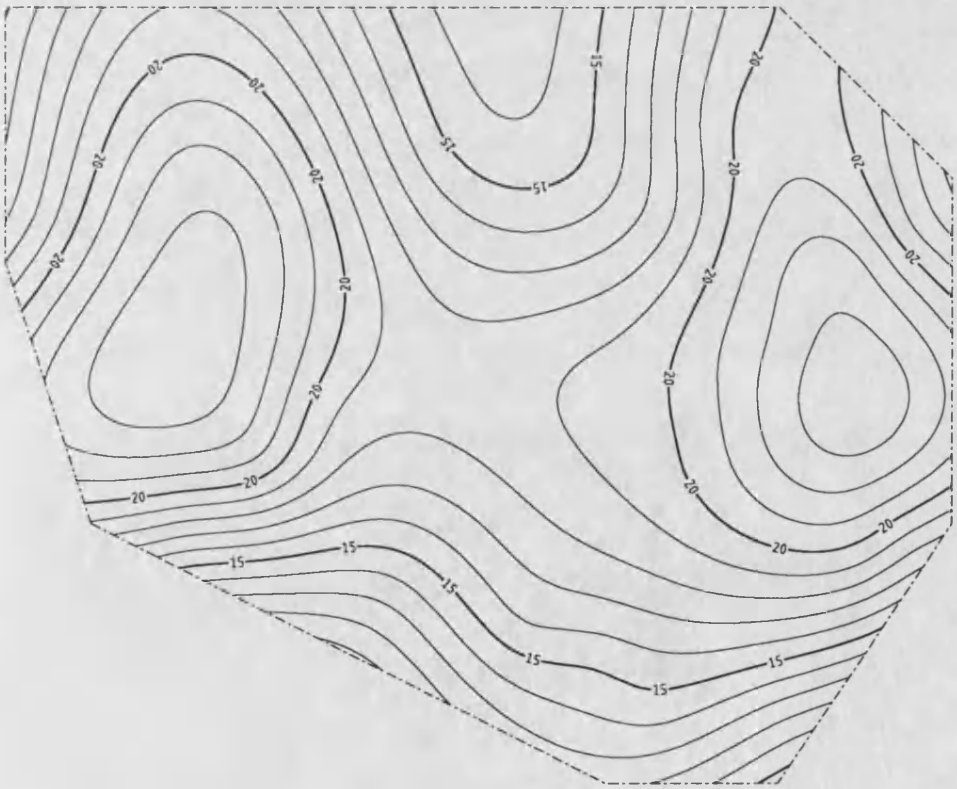


fig 3.4.8 Error matched

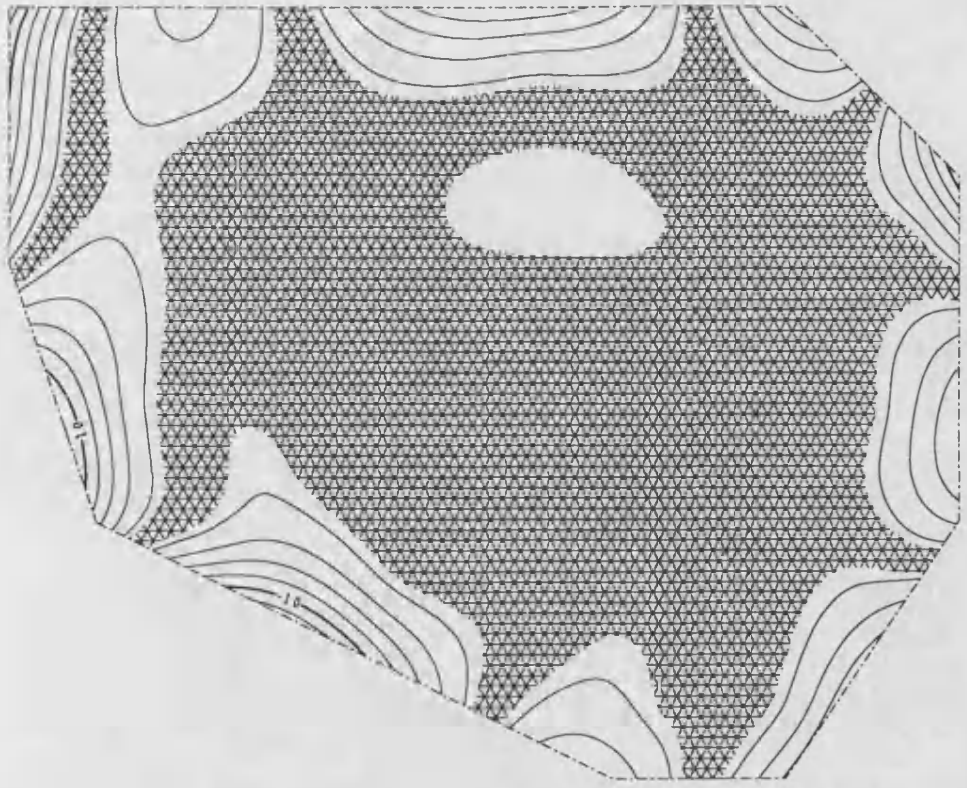


fig 3.4.9 Difference between thin-plate and error matched

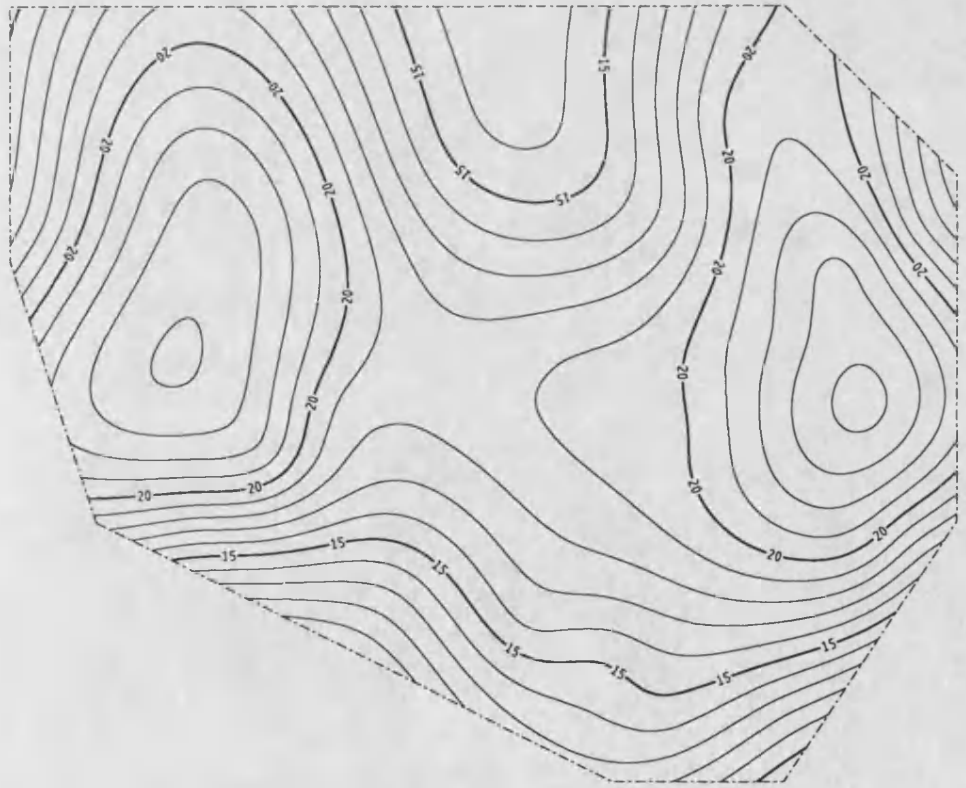


fig 3.4.10 Roughness matched

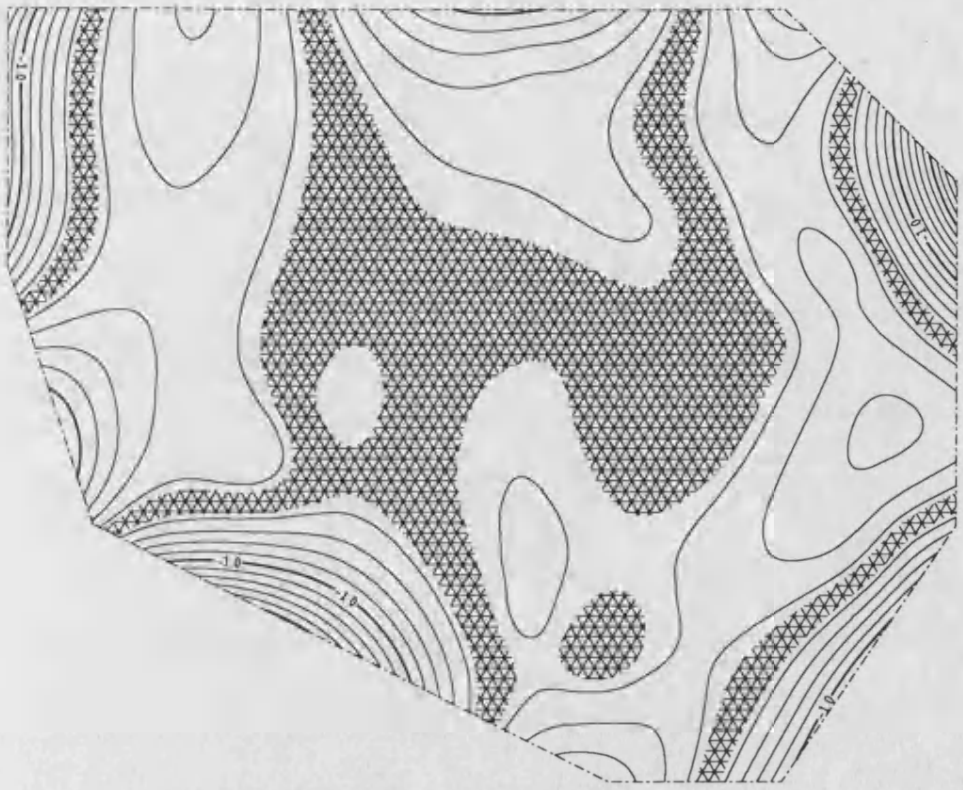


fig 3.3.11 Difference between thin-plate and roughness matched

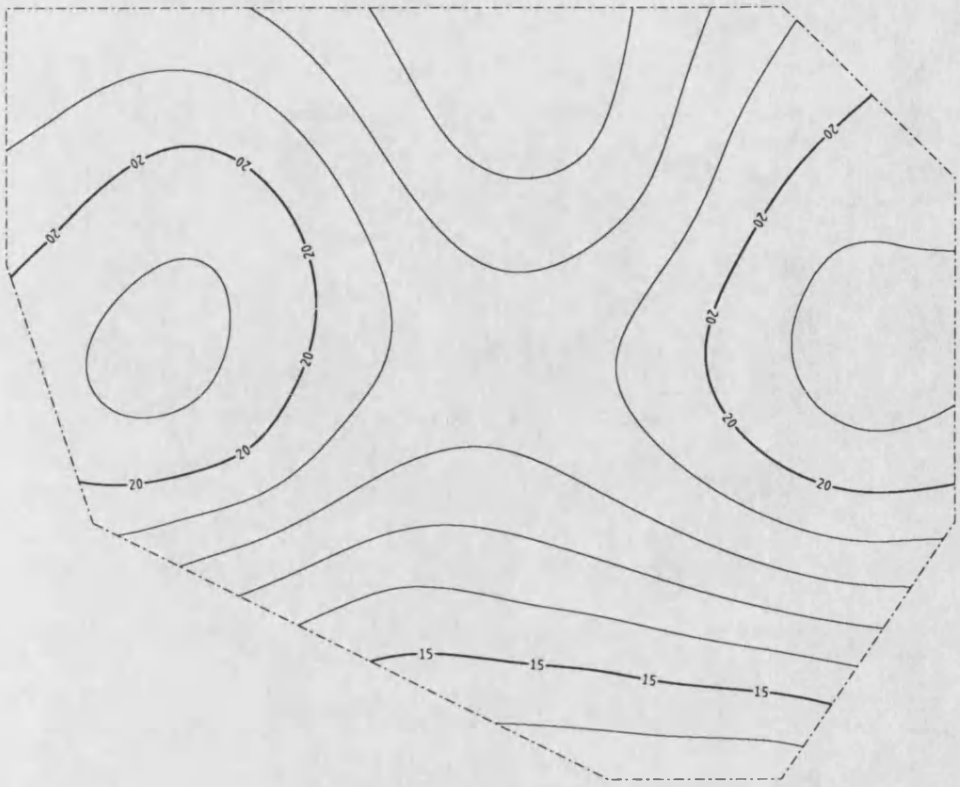


fig 3.4.12 Thin-plate smoother (alpha=100)

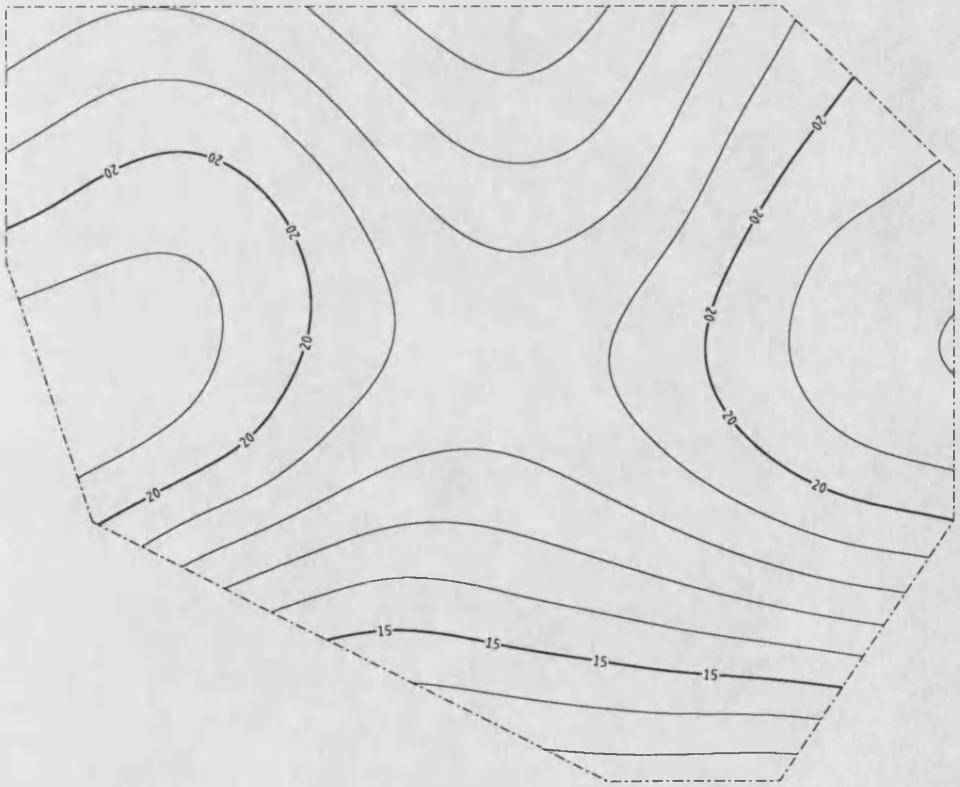


fig 3.4.13 Error matched

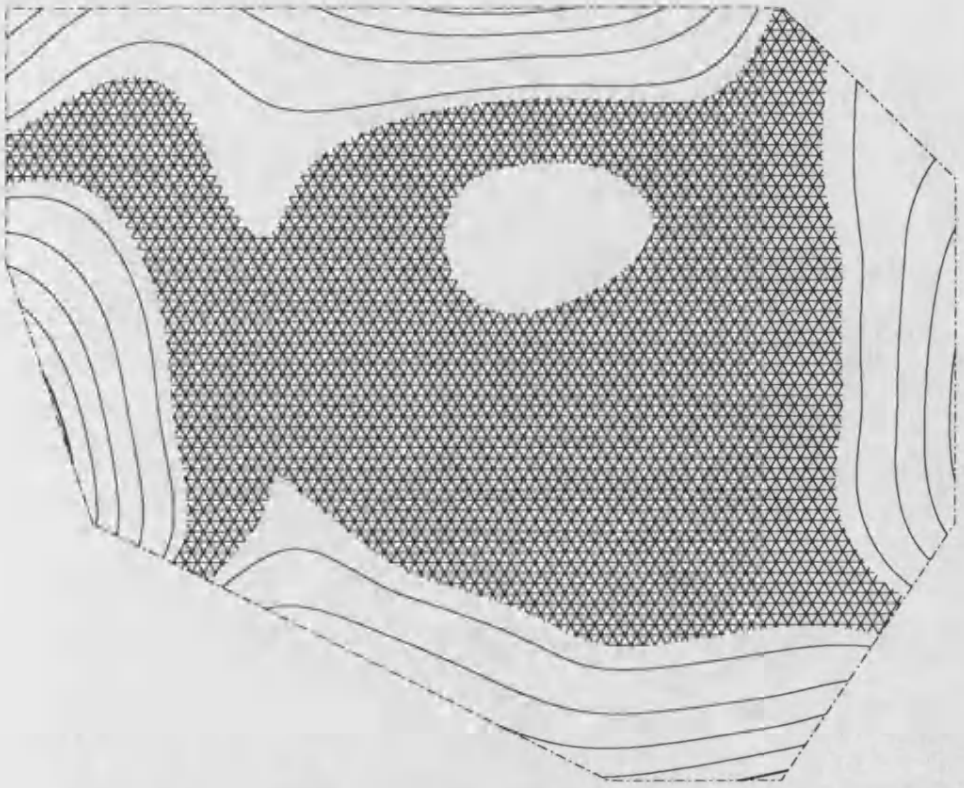


fig 3.4.14 Difference between thin-plate and error matched

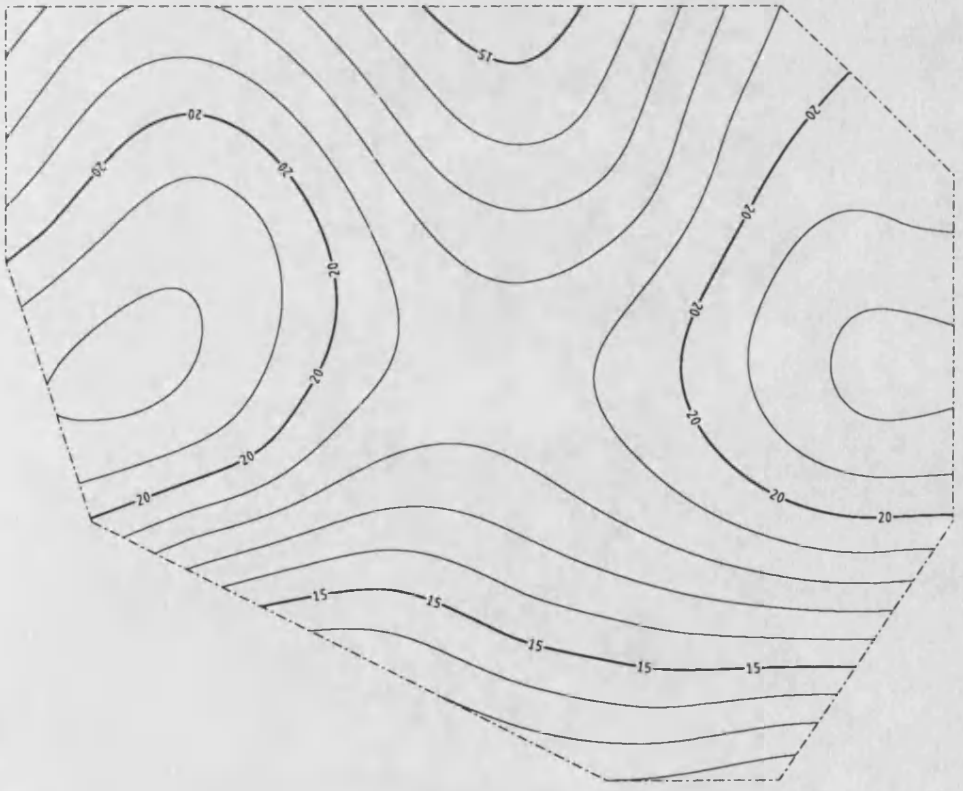


fig 3.4.15 Roughness matched

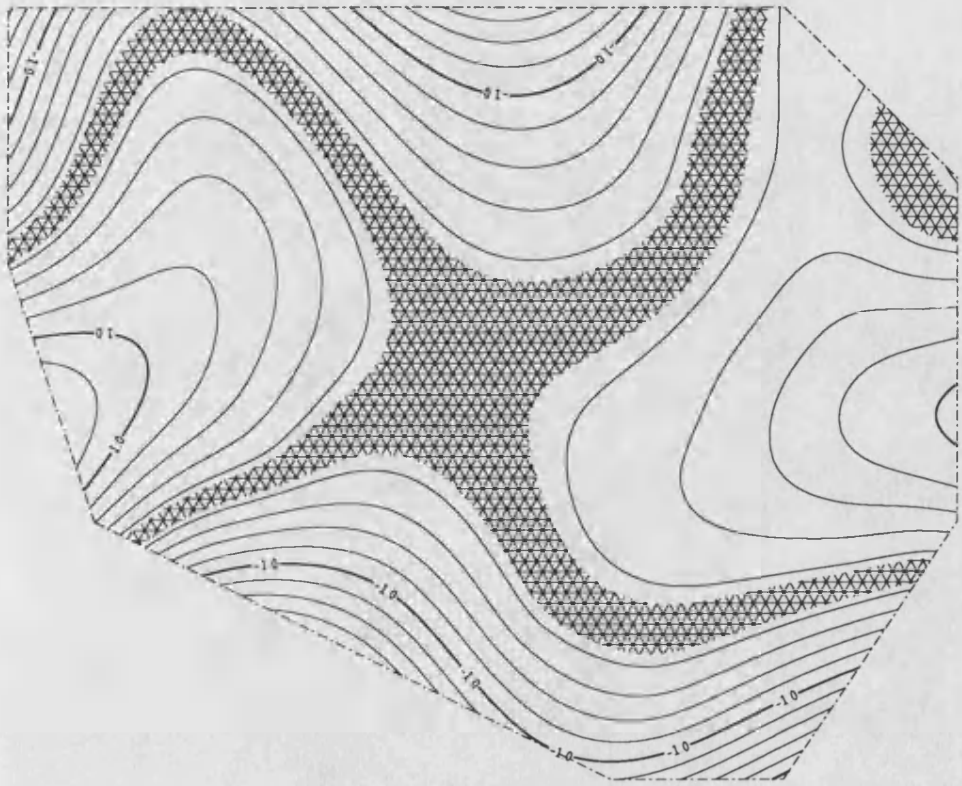


fig 3.3.16 Difference between thin-plate and roughness matched

that since the solution with an infinite amount of smoothing is the same plane for both types of spline that the differences are zero as the smoothing parameter tends to infinity. Thus as we increase the amount of smoothing away from interpolation, it seems that in this case at least, the differences between the error matched splines first decrease slightly, then increase more significantly and then at some point begin to tend to zero. The largest differences occur, in this data set, between the roughness matched splines, when we have smoothed considerably, but are still extracting non-linear trend information from the data.

Although this section looked only at the first data set, similar results apply to the other data sets we have looked at. The slight decrease in the differences between the error matched splines at the lowest level of smoothing, over the interpolating splines, does not occur consistently, and may be a problem associated with the actual matching of the error penalties. More accurate matching, than to two decimal places, may have eliminated this decrease. In any case the reduction is very slight, and could easily be attributed to numerical errors, either in the spline procedures or the contouring package. The difference surface is very flat in the centre of the data and it is known that this makes the contours less well determined.

Chapter 4.
Natural Neighbour Splines.
4.1. The Natural Neighbour Spline Problem.
4.1.1. A Discrete Roughness Penalty in One Dimension.

A slightly different approach to bivariate (and higher dimensional) splines, arises from an alternative view of univariate splines. The univariate smoothing spline is characterised as the solution of the variational problem,

$$\text{minimise} \sum_{i=1}^{N}(z_i - f(t_i))^2 + \alpha\int_a^b f''^2$$

where $a<t_i \ldots <t_N<b$. As we have seen in the earlier chapters the error term carries straight over to higher dimensions without modification. So it is the roughness term which causes difficulty. The spline approach discussed earlier generalises this integral to an integral over some region Ω of the sum of all the second derivatives squared (including $\partial^2 f/\partial x\partial y$ and $\partial^2 f/\partial y\partial x$), and proceeds to solve the minimisation problem using functional analysis.

If we notice, however, that in one dimension when the function f is cubic in the interval (t_i,t_{i+1}) where $1\leq i<N$, as it is in the case of a univariate spline, the contribution to the roughness of this section of the spline can be written,

$$R_i = \frac{(g_{i+1}-g_i)^2}{(t_{i+1}-t_i)} + \frac{3}{(t_{i+1}-t_i)^3}[(g_{i+1}+g_i)(t_{i+1}-t_i)-2(z_{i+1}-z_i)]^2$$

where $z_i=f(t_i)$ and $g_i=f'(t_i)$ for all i (Sibson (1985)).

Accordingly this expression can be regarded as a discrete approximation of the contribution in the interval (t_i,t_{i+1}) to the roughness of any function f, the approximation being exact whenever f is cubic in this interval. Natural neighbour splines arise from a generalisation of this *finite difference* approximation to higher dimensions.

Natural neighbour splines also rely on the observation that many smoothing techniques use a two stage approach. That is firstly the data itself is smoothed to obtained smoothed values at the data sites and secondly an interpolant is fitted to these smoothed values. It is the first of these stages that will be considered in this part of this thesis. For a detailed discussion of the second stage see Sibson (1985).

In fact the natural neighbour approach has a first stage that estimates smoothed values and gradients at the data sites (only gradients in the case of interpolation) and then uses a similar procedure to interpolate between the data sites. We shall

now describe the details of natural neighbour splines which are relevant to the first of the above mentioned stages, the implementation of which is our concern here.

In the case of univariate splines, we have seen that the roughness of the *spline* can be written as,

$$R_{total} = \sum_{i=1}^{N-1} R_i$$

where R_i was described above. Notice that the term R_i involves only information at the points t_i and t_{i+1}, and R_i can be regarded as the contribution to the roughness of the function of the link between this pair of neighbours. The total roughness R_{total} is then seen to be a sum of contributions over neighbour pairs, counting each pair only once.

Thus before we can generalise R_i to two dimensions we must first generalise the concept of a neighbour.

4.1.2. The Dirichlet Tessellation.

We shall use the well known method of determining neighbours of a point via the Dirichlet tessellation. Firstly we describe the tessellation in its k-dimensional setting.

Suppose we have N points in $\mathbf{R}^k$, $t_1,\ldots,t_N$ and a *convex* region $\Omega \subset \mathbf{R}^k$, containing all the data sites. In fact we assume Ω to be an open, bounded polyhedron, as in the chapters on finite window splines, although we did not require Ω to be convex there. For each point t_i define,

$$T_i = \{\, t \in \Omega : \|t_i - t\| < \|t_j - t\| \quad \forall\ j \neq i \,\}$$

where $\|\cdot\|$ is the usual Euclidean distance norm in $\mathbf{R}^k$. Since T_i is just the intersection with the window Ω of $N-1$ half spaces, it is itself a convex polyhedron. T_i is in fact that part of the window which is closer to t_i than any other t_j for $j \neq i$, and we can see that,

$$\bigcup_{i=1}^{N} T_i \subset \Omega \subset \bigcup_{i=1}^{N} \bar{T}_i.$$

That is, Ω is covered by the union of the T_i up to a null set.

T_i is called the *tile*, or *Voronoi* or *Thiessen polyhedron* of t_i and the collection of tiles is called the Dirichlet tessellation of Ω defined by $t_1,\ldots,t_N$. Two tiles in the tessellation are said to be contiguous if they share a common boundary facet ($k-1$ dimensional) and we define the relation *is a neighbour of* by,

t_i is a neighbour of $t_j \iff T_i$ and T_j are contiguous.

This gives us a neighbourhood system in the qualitative sense, but we will require a quantitative measure of the *strength* of the neighbour relation. In one dimension this is provided by the reciprocal of the distance between two neighbours, which is sufficient, since two neighbours do not have any other points between them, and the neighbours of a point are just those closest to it on either side. In higher dimensions however this is not sufficient. Consider the points below,

t_4 ·

t_3 · t_1 · t_2 ·

t_5 ·

In this case, although the two points t_2, t_3 are about the same distance from t_1, t_2 plays a much more important role than t_3 because of the positions of t_4 and t_5. In fact in this case t_2 provides the only information to the right of t_1. To quantify this importance we introduce the so called *boundary over distance* weights.

4.1.3. Boundary over Distance Weights.

Define $\forall\ i \neq j$,

$$S_{ij} = \{\ t \in \Omega : \|t_i - t\| = \|t_j - t\| < \|t_l - t\| \quad \forall\ l \neq i,j\ \}$$

Then S_{ij} is the $k-1$ dimensional common facet of tiles T_i and T_j. Using σ_{k-1} to mean $k-1$ dimensional Lebesgue measure put,

$$v_{ij} = \sigma_{k-1}(S_{ij}) / \|t_i - t_j\|.$$

v_{ij} is called the boundary over distance weight or BOD weight. Notice that for $k=1$ S_{ij} is a point midway between t_i and t_j, and thus $v_{ij} \equiv 1/|t_i - t_j|$, if i and j are neighbours. This means that the reciprocal of the distance between two neighbours is just the one dimensional BOD weight.

An important property of the v_{ij}, proved in Sibson (1980) and used extensively in Sibson (1985) is called the local coordinates property. This states that, given that t_i is not neighbouring the boundary we have,

$$\sum_{j \neq i} v_{ij}(t_j - t_i) = 0.$$

Recall that in one dimension the roughness was written, using the new notation,

$$\sum_{pairs\ i\neq j} v_{ij}\{(g_j-g_i)^2+\frac{3}{|t_j-t_i|^2}[(g_j+g_i)(t_j-t_i)-2(z_j-z_i)]^2\}$$

where here the sum is taken over unordered pairs with $i\neq j$ and v_{ij} is zero if i and j are not neighbours.

4.1.4. A Bivariate Generalisation of the Discrete Roughness Penalty.

We shall now generalise the above univariate penalty to two dimensions. Since the second term in the summand vanishes whenever the z_i, z_j, g_i and g_j lie on a quadratic rather than a general cubic, we consider the first term, separately.

The roughness reduces to,

$$\sum_{pairs\ i\neq j} v_{ij}(g_j-g_i)^2$$

(where the is taken over the pairs mentioned above) and this correctly recovers the roughness of a C^1 piecewise quadratic (linear outside the range of the data). This sum can easily be generalised to higher dimensions by replacing the squared change in derivative between t_i and t_j with the squared norm of the change in derivative, as below.

$$\sum_{pairs\ i\neq j} v_{ij}(g_j-g_i)^{\mathrm{T}}(g_j-g_i)$$

and in fact this recovers the roughness of a general quadratic in two dimensions up to an edge effect near the window boundary.

In this way Sibson (1985) proposes that the form,

$$R_G=\sum_{pairs\ i\neq j} v_{ij}\left\{(g_j-g_i)^{\mathrm{T}}(g_j-g_i)+\frac{\theta_k}{\|t_j-t_i\|^2}[(g_j+g_i)^{\mathrm{T}}(t_j-t_i)-2(z_j-z_i)]^2\right\}$$

should be used as the appropriate generalisation of the discrete roughness penalty, where θ_k is a dimension dependent constant. For $k=1$ $\theta_k=3$ is the appropriate choice as we have seen and Sibson (1985) has an argument that suggests $\theta_k=k+2$ is the best choice that can be made for θ_k. This choice does not make R_G recover the roughness (even up to edge effects) of any larger set of functions than general quadratics, but it is the preferred choice within this form of R_G. More complicated forms of R_G are not considered here.

4.1.5. Statement of the Natural Neighbour Spline Problem.

We can now state our problem, which corresponds to the first stage of the natural neighbour spline fitting problem, namely determining fitted values and gradients at the data sites.

Problem. Given data sites $t_1,\ldots,t_N \in \Omega \subset \mathbf{R}^2$, where Ω is a convex polygonal region, and corresponding data values $Z_1,\ldots,Z_N$, find,

(interpolation) $g_1,\ldots,g_N$ such that,

$$R_G = \sum_{pairs\ i\neq j} v_{ij}\left\{(g_j-g_i)^{\mathrm{T}}(g_j-g_i) + \frac{\theta_2}{\|t_j-t_i\|^2}[(g_j+g_i)^{\mathrm{T}}(t_j-t_i)-2(Z_j-Z_i)]^2\right\}$$

is a minimum, or

(smoothing) $z_1,\ldots,z_N$ and $g_1,\ldots,g_N$ such that,

$$\sum_{i=1}^{N}(Z_i-z_i)^2+\alpha \sum_{pairs\ i\neq j} v_{ij}\left\{(g_j-g_i)^{\mathrm{T}}(g_j-g_i) + \frac{\theta_2}{\|t_j-t_i\|^2}[(g_j+g_i)^{\mathrm{T}}(t_j-t_i)-2(z_j-z_i)]^2\right\}$$

is a minimum.

Hence both the interpolation and smoothing problems reduce (at least in their first stage) to minimising a quadratic form in $2N$ and $3N$ variables respectively. This corresponds to the inversion of $2N$ and $3N$ matrices, which are approximately two and three times the size of the matrices to be inverted when using thin plate splines. Thus we might expect natural neighbour splines to pose severe computational problems, particularly for large data sets.

However, returning to the quadratic forms it is easy to see that the variables associated with each data site t_i, namely z_i and g_i, occur in terms in the sums only with the variables associated with the neighbours of t_i. In terms of the matrices to be inverted this means that in each row, row i say, the only entries which are non-zero are the entries corresponding with the variables associated with t_i and the neighbours of t_i, and since the number of neighbours of a point is relatively small, the matrices are very sparse. In fact, in two dimensions, it is known that each point has on average six neighbours, so that the average number of non-zero entries in each row does not increase with the number of data sites, it remains at six times the number variables associated with each data site plus the number of variables associated with t_i itself. Thus on average 14 or 21 non-zero entries in each row can be expected.

4.2. The Preconditioned Conjugate Gradients Method for Inverting a Sparse Matrix.

In the previous section we have seen that in order to determine fitted values and gradients at the data sites to fit natural neighbour splines, we need to solve a possibly very large, sparse linear system. Duff (1982) presents a recent review of sparse matrix techniques. For our problem we have chosen, firstly, to use an iterative technique known as the conjugate gradients method. (see Reid (1971)).

4.2.1. The Conjugate Gradients Algorithm.

The conjugate gradients method was discussed by Hestenes and Stiefel (1952), as an n-step procedure for solving an $n \times n$ linear system having a symmetric positive definite matrix of coefficients. In this form it is not regarded as an iterative procedure, but rather as a terminating direct method.

The method regards solving the set of equations,

$$Ax = b$$

as equivalent to minimising the quadratic form,

$$Q(x) = \tfrac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b$$

where A is an $n \times n$ positive definite symmetric matrix and b is an n vector of right hand sides. In order to minimise Q, n search directions are set up $d_1, \ldots, d_n$ and exact line search is used along each direction in turn starting from an initial approximation, the solution of one line search being used as the initial approximation for the next search. The search directions used are designed to be conjugate, that is they are chosen so that,

$$d_i^{\mathrm{T}} A d_j = 0 \quad i \neq j.$$

In this case it can be shown, (Reid (1971)), that if exact arithmetic were in use, the exact solution would be found after $m \leq n$ searches. Normally $m=n$, but repeated eigenvalues in A or a fortunate choice of initial approximation can lead to $m<n$. Unfortunately, however, in the presence of rounding error this property no longer holds, and in any case as a direct method, the conjugate gradients algorithm is not competitive in storage or number of floating point operations, with Gaussian elimination.

A certain choice of d_i, see below, leads to the property that the solution of the exact line search may be very close to the solution of the linear system after many fewer than n-steps. This means that the method is very useful as an iterative technique. We use the algorithm in its form presented by Reid (1971) and Axelsson (1976), and given below.

$$
\begin{aligned}
x_0 &= \text{initial approximation} \\
r_0 &= Ax_0 - b \\
d_0 &= -r_0 \\
\text{repeat for} &\quad i=0,1,2,\ldots \\
\lambda_i &= r_i^{\mathrm{T}} r_i / d_i^{\mathrm{T}} A d_i \\
x_{i+1} &= x_i + \lambda_i d_i \\
r_{i+1} &= r_i + \lambda_i A d_i \\
\beta_i &= r_{i+1}^{\mathrm{T}} r_{i+1} / r_i^{\mathrm{T}} r_i \\
d_{i+1} &= -r_{i+1} + \beta_i d_i \\
\text{until} &\quad r_{i+1}^{\mathrm{T}} r_{i+1} < \varepsilon.
\end{aligned}
$$

Notice the use of a recursive formula for calculating the residuals at each iteration which does not entail the extra matrix multiplication required by the alternative formula,

$$r_{i+1} = Ax_{i+1} - b.$$

Reid (1971) finds in his experiments that the two formulae give closely consistent results. At each stage, therefore, only one matrix multiplication is necessary to find Ad_i and since A is sparse this can be a not too expensive operation. Because of this, used as an iterative technique, the conjugate gradients method can compete very well with direct methods for inverting sparse matrices.

The above choice for the search directions d_i ensures that $d_i^{\mathrm{T}} A d_j = 0 \ \forall i \neq j$ as is shown in Reid (1971), and so the search directions are conjugate (if exact arithmetic were being used). The inexact nature of numerical computation, however, implies that the search directions will only be approximately conjugate and since at each stage we multiply this direction by the matrix A, the growth of errors in the search directions, and hence the other vectors, is dependent on the conditioning of the matrix A. Reid (1971) makes a comparison of the conjugate gradients method with two other iterative techniques, and concludes that the method competes favourably when A is a well-conditioned matrix.

4.2.2. The Preconditioned Algorithm.

In our problem, since the matrix involved is derived from a discrete approximation to the thin plate spline problem, it may be the case that our matrix shares the ill-conditioning of the thin plate problem. In fact this is the case and although sparse, the matrices derived from the natural neighbour spline problem can be quite badly conditioned, in the spectral condition number sense, particularly when smoothing, as we shall see later.

It can be shown (see Axelsson (1976)) in fact that the rate of convergence of the conjugate gradients method depends upon this spectral condition number and we shall see that for our problem, convergence of the above algorithm can be quite slow. The rate of convergence can however be improved by the technique of preconditioning. This involves finding a matrix E such that the transformed system of equations,

$$(E^{-1}AE^{-\mathrm{T}})(E^{\mathrm{T}}x) = (E^{-1}b)$$

(where here $E^{-\mathrm{T}}$ denotes $(E^{-1})^{\mathrm{T}}$) has a much better conditioned matrix than the untransformed system. In principle the transformed system is then solved and the untransformed solution recovered.

This method may have the serious draw back that $E^{-1}AE^{-\mathrm{T}}$ may not be as sparse as A, and then the algorithm will require many more operations per iteration to perform the matrix multiplication, even though it takes fewer iterations. This may mean there is not actually a reduction in work. Also the matrix E must be easily invertible so that $E^{-1}b$ and $E^{-1}AE^{-\mathrm{T}}$ can be easily calculated at little cost.

The difficulty with loss of sparseness can be overcome by transforming the conjugate gradients algorithm to form a preconditioned algorithm. Axelsson (1976) gives the following form of the algorithm, and it is this form that we shall use.

$$\begin{aligned}
x_0 &= \text{initial approximation} \\
r_0 &= Ax_0 - b \\
\text{solve } C\gamma_0 &= r_0 \\
d_0 &= -\gamma_0 \\
\text{repeat for} &\quad i=0,1,2,\ldots \\
\lambda_i &= r_i^{\mathrm{T}}\gamma_i / d_i^{\mathrm{T}}Ad_i \\
x_{i+1} &= x_i + \lambda_i d_i \\
r_{i+1} &= r_i + \lambda_i Ad_i \\
\text{solve } C\gamma_{i+1} &= r_{i+1} \\
\beta_i &= r_{i+1}^{\mathrm{T}}\gamma_{i+1} / r_i^{\mathrm{T}}\gamma_i \\
d_{i+1} &= -\gamma_{i+1} + \beta_i d_i \\
\text{until} &\quad r_{i+1}^{\mathrm{T}}\gamma_{i+1} < \varepsilon.
\end{aligned}$$

where $C = EE^{\mathrm{T}}$.

This algorithm uses the same matrix multiplication as the previous unpreconditioned algorithm, and adds an extra matrix step, namely, solve $C\gamma_{i+1} = r_{i+1}$ at each iteration. However this algorithm is equivalent to the unpreconditioned algorithm replacing A by $E^{-1}AE^{-\mathrm{T}}$ etc. This matrix is better

conditioned (by choice of E) and so convergence should be faster. We do not require E in the preconditioned algorithm, only $C=EE^{\mathrm{T}}$ and since $C^{-1}A$ is similar to $E^{-1}AE^{-\mathrm{T}}$ and therefore has the same condition number, we can choose C immediately without determining E. C is chosen so that $C^{-1}A$ has a smaller condition number than A. Again the technique fails to improve the overall work rate of the algorithm if C is not much easier to invert than A, since we must invert C at each iteration. However if we choose C to be sufficiently easy to invert this will not slow the algorithm significantly at each iteration, and fewer iterations can be expected which means the algorithm will be faster. C is called the preconditioning matrix.

4.2.3. Some Preconditioning Techniques.

The choice of preconditioning technique corresponds to the choice of matrix C. The simplest technique is that of diagonal scaling. In this case we set,

$$C=\mathrm{diag}(a_{11},\ldots,a_{nn}) \quad \text{where} \quad \underset{n\times n}{A}=(a_{ij}).$$

This C is positive definite since A is and we have $a_{ii}=e_i^{\mathrm{T}}Ae_i>0$ where e_i is the ith standard basis vector. Also C is trivial to invert, being a diagonal matrix so very little extra work is required in the preconditioned algorithm. The naive nature of this preconditioning means that the spectral condition number may not be reduced sufficiently to reduce the number of required iterations significantly.

A more sophisticated approach is provided by the technique of Symmetric Successive Over-Relaxation or SSOR. SSOR is an iterative symmetric matrix inversion method in its own right (see Young (1971)), and SSOR preconditioned conjugate gradients can also be regarded as SSOR inversion with conjugate gradients applied as an acceleration method. In this case we choose,

$$C=(D+\omega L)D^{-1}(D+\omega L^{\mathrm{T}})$$

where here $A=L+D+L^{\mathrm{T}}$ with D being the diagonal part of A and L the strictly lower triangular part. ω is an over-relaxation parameter which must lie between 0 and 2 for convergence of the algorithm to be assured. Notice that $\omega=0$ corresponds to the diagonal scaling technique.

The choice of ω is by no means a trivial problem, and its theoretically optimal value, that which reduces the spectral condition number of $C^{-1}A$ the most, is given by,

$$\omega_{opt} = \frac{2}{1+2\sqrt{(\frac{1}{2}+\delta)/\mu}} \quad \text{where} \quad \mu = \max_x \frac{x^T D x}{x^T A x}$$

$$\text{and} \quad \delta = \min\{0, \max_x \frac{x^T [L D^{-1} L^T - \frac{1}{4} D] x}{x^T A x}\}$$

see Axelsson (1976).

This optimal value is usually too expensive to calculate and estimated values must be used. Axelsson however points out that the actual number of iterations of the SSOR preconditioned conjugate gradients algorithm required to reach a specified accuracy is quite insensitive to the choice of ω, so long as it is near ω_{opt}. In using this method for our problem an estimate of ω_{opt} was obtained by trying several values of ω. This was repeated for differing interpolation and smoothing problems, and the choice of ω which produced the minimum number of iterations was subsequently used in the other examples. In fact in all cases this estimated value for ω_{opt} was observed to be close to 1.0 and so we replaced ω by this value in our implementation, of SSOR preconditioned conjugate gradients.

The third, and most complex, form of preconditioning we have used is incomplete Cholesky decomposition. It is well known that any positive definite matrix can be written in the form $A = LL^T$, where L is some lower triangular matrix. Unfortunately finding the factor L is a major part of the work involved in a direct method of inverting A, and it is known that L may contain many more non-zero elements than A. Since for our problem A is very sparse we do not want to lose the sparsity of the problem in the inversion procedure. In fact the formulation of the natural neighbour spline problem as presented here, was designed to generate this sparsity.

We can preserve some of this sparsity by finding a sparse matrix L such that, $C = LL^T$ has the property that $C^{-1}A$ is better conditioned than A, rather than $C^{-1}A = I$. In this case L is found so that $LL^T = A+E$ where E is an error matrix, and in some sense small. This is the basis of incomplete Cholesky decomposition.

In fact it is often advantageous to permute the rows and columns of A (equivalent to reordering the data sites) before an L is chosen, in order to reduce the approximation errors E. Thus we actually have,

$$LL^T = PAP^T + E$$

where P is a permutation matrix. We assume in what follows that a suitable reordering of the rows and columns of a has occurred. Such an ordering will be suggested shortly.

The Cholesky decomposition can be written in a square root free form, namely, $A = LDL^{\mathrm{T}}$ where D is a diagonal matrix and L is lower triangular, both given by the algorithm,

$$\begin{aligned}
&\text{for } i = 1,\ldots,n \\
&\quad \text{for } j = i, i+1,\ldots,n \\
&\qquad L_{ji} = A_{ji} - \sum_{k=1}^{i-1} L_{jk} L_{ik} D_{kk} \\
&\quad D_{ii} = (L_{ii})^{-1}
\end{aligned}$$

However as pointed out above, the L determined in this way has many more non-zero entries than A. In fact it can be seen that L can have non-zero entries in row j from element L_{jm} to element L_{jj} where m is such that A_{jm} is the first non-zero entry in row j of A. In this way it is possible for the Cholesky decomposition of A to have *filled-in* the *envelope* of the matrix. If the data sites are ordered in such a way that the non-zeros in A are close to the diagonal, the fill-ins are kept to a fewer number, but this may still be too many.

The above ordering corresponds, in our problem, to sorting the data sites so that neighbours occur only close to each other in the reordered collection of sites. Almost equivalently, the sites are reordered so that sites close to each other in the plane are close to each other in the ordering.

For our purposes, we do not require the Cholesky decomposition, only some matrix C such that $C^{-1}A$ is well-conditioned. The suggested technique is therefore to fix some entries of L, the Cholesky factor, to be zero, regardless of whether they should fill-in in the decomposition. The above algorithm is used, but the chosen entries are fixed to be zero, and the algorithm proceeds with calculating the other entries, using zero for the entries so fixed. In this way a sparse approximation to L can be obtained, (say $\tilde{L}$), and the corresponding D, ($\tilde{D}$), found so that $C = \tilde{L}\tilde{D}\tilde{L}^{\mathrm{T}}$ can be used as a preconditioning matrix. As this matrix is composed of sparse matrices, it can be stored cheaply as its factors, and it is easily inverted since all its factors are easily inverted. Clearly the reduction in spectral condition number obtained will depend on how many fill-ins are allowed, with all fill-ins permitted resulting in a condition number of 1, (at least in theory.)

The decomposition $\tilde{L}\tilde{D}\tilde{L}^{\mathrm{T}}$ is known as an incomplete Cholesky factorisation of A. There are at least two approaches to the problem of the choice of the elements which are to be allowed to fill-in. One procedure is to allow fill-ins wherever the value of the filling element exceeds a certain threshold. Munksgaard (1980) uses this technique, allowing an element L_{ji} to fill-in whenever $L_{ji} > c\sqrt{(L_{ii}A_{jj})}$ where c is a positive parameter, dependent on how sparse the factors are required to be.

An alternative approach, is to force a predetermined sparsity pattern onto L. Kershaw (1978) discusses this method, and he uses the simplest of this group of techniques, namely, that where the sparsity pattern enforced is that of the matrix A. For our investigations we shall use this method. It has the advantage that $\tilde{L}$ is known to be as sparse as A before we start to calculate it, and therefore requires only as much storage as A does (A is symmetric, L lower triangular).

There remains one final problem. Since the elements of $\tilde{L}$ are in error whenever a fill-in is suppressed, these errors will propagate into other columns of the decomposition, causing even those elements allowed to fill-in to be incorrect, that is not the same as the corresponding elements in the complete factorisation. Since C is only to be used as a preconditioning matrix this is not important for the elements in $\tilde{L}$, but if an element in $\tilde{D}$ becomes negative (in error) then $C=\tilde{L}\tilde{D}\tilde{L}^{\mathrm{T}}$ will cease to be positive definite, and so the conjugate gradients algorithm may not be convergent. Both Munksgaard (1980) and Kershaw (1978) address this problem, and suggest different solutions. Munksgaard suggests multiplying the diagonal elements of A by a scalar $(1+\eta)$ where $\eta>0$, before the start of the incomplete factorisation algorithm. η must be chosen so that no D_{ii} becomes negative. Kershaw remarks that since the factorisation is only meant to be approximate, we can simply set the negative D_{ii} to some suitable positive value as they occur.

In our experience, the first method usually results in the smaller number of iterations, with a sensible choice for η. (We have used $\eta\equiv 1/100N$ in the computations which follow.) In fact for certain forms of matrices it can be proved that $\eta\equiv 0$ will do (Kershaw (1978)) and for our matrices, provided a good ordering of the data sites is found, this value is satisfactory in many cases. However some non-zero value of η is required, in some cases, particularly when A is very ill-conditioned (see later).

Before we go on to discuss our implementations of the above techniques for the natural neighbour spline problem, there is a group of preconditioning methods not mentioned above, which are applicable to our problems. Remember that each data site has 2 (interpolation) or 3 (smoothing) unknowns associated with it, in the quadratic from to be minimised. These are namely the fitted gradients or fitted values and gradients at the data site. The sparsity pattern of the derived matrix depends only on the neighbourhood relation, which depends on the positions of the data sites, so that the matrix can be thought of as an $N\times N$ matrix of 2×2 or 3×3 blocks, having the same sparsity pattern in both cases. Each diagonal block relates the variables of one data site to themselves, and each non-zero, off-diagonal block relates the variables at one data site to the variables of one of its

neighbours. This partitioning of the matrix gives rise to the block preconditioning techniques (mentioned briefly by Axelsson (1976)).

The simplest is block diagonal scaling, where the preconditioning matrix is taken to be the matrix composed of the diagonal blocks of A. This matrix is easy to invert as it is only necessary to invert N 2×2 or 3×3 matrices. The second block-approach preconditioning we have looked at is block SSOR. In this technique the matrices in the element-wise formulation of the preconditioning matrix, $C=(D+\omega L)D^{-1}(D+\omega L)^{\mathrm{T}}$, are replaced by their block equivalents, namely the block diagonal matrix and the block lower triangular matrix.

The computation of the block incomplete Cholesky factorisation is somewhat complicated, not least because the problem of diagonal entries becoming negative is replaced by the harder to detect and solve problem of diagonal block matrices becoming non-positive definite. For this reason, and because the results suggest it is unnecessary, we have not looked at this preconditioning technique in this thesis.

4.3. The Implementation of Preconditioned Conjugate Gradients Method for the Natural Neighbour Spline Matrices.

The use of conjugate gradients to solve our problem requires that the matrices to be inverted are positive definite. Before we discuss the implementation of the techniques mentioned above, therefore, we first establish their applicability.

4.3.1. The Positive Definiteness of the Natural Neighbour Spline Matrices.

We deal first with the interpolation case. Remember that the quadratic form to be minimised can be written,

$$R_G = \tfrac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b + \text{constant}$$

where R_G is defined in section 4.1.5 and we define,

$$\underset{2N\times 1}{x} = \begin{bmatrix} g_1 \\ \vdots \\ g_N \end{bmatrix}$$

where each g_i is a 2×1 vector of gradients at data sites t_i. A and b are also defined accordingly.

Thus comparing the purely quadratic portions of both sides we see that,

$$x^{\mathrm{T}}Ax = \sum_{i=1}^{N}\sum_{j\neq i} v_{ij}\left\{\|g_j-g_i\|^2 + \frac{\theta_2}{\|t_j-t_i\|^2}[(g_j+g_i)^{\mathrm{T}}(t_j-t_i)]^2\right\}$$

To prove positive definiteness of A, the matrix to inverted in the interpolation problem, consider x such that $x^{\mathrm{T}}Ax = 0$. This implies that,

$$v_{ij}\{\|g_j-g_i\|^2 + \frac{\theta_2}{\|t_j-t_i\|^2}[(g_j+g_i)^{\mathrm{T}}(t_j-t_i)]^2\} = 0$$

for each i and j with $i\neq j$, since this summand is at least non-negative. However v_{ij} is non-zero whenever i and j are neighbours, so that we have,

$$\begin{aligned} \|g_j-g_i\|^2 = 0 \quad & \text{whenever } i \text{ and } j \text{ are neighbours,} \\ \Rightarrow \quad g_i = g_j \quad & \text{whenever } i \text{ and } j \text{ are neighbours,} \\ \Rightarrow \quad g_i = g \quad & \text{(say) } \forall i \text{ provided } \Omega \text{ is connected.} \end{aligned}$$

and then the second term gives,

$$\begin{aligned} [(g_i+g_j)^{\mathrm{T}}(t_j-t_i)]^2 = 0 \quad & \text{whenever } i \text{ and } j \text{ are neighbours,} \\ \Rightarrow \quad g^{\mathrm{T}}(t_j-t_i) = 0 \quad & i \text{ and } j \text{ neighbours,} \\ \Rightarrow \quad g \equiv 0 \quad & \text{provided } t_i \text{ not all colinear.} \end{aligned}$$

Therefore $x\equiv 0$, that is A is positive definite.

In the smoothing case the purely quadratic part of the form to be minimised corresponds to,

$$\sum_{i=1}^{N} z_i^2 + \alpha \sum_{i=1}^{N} \tfrac{1}{2}\sum_{j\neq i} v_{ij}\left\{\|g_j-g_i\|^2 + \frac{\theta_2}{\|t_j-t_i\|^2}[(g_j+g_i)^{\mathrm{T}}(t_j-t_i)-2(z_j-z_i)]^2\right\}$$

Setting this equal to zero, and noting that both sums are at least non-negative (provided that $\alpha\neq 0$) we see that,

$$\sum_{i=1}^{N} z_i^2 = 0 \quad \Rightarrow \quad z_i = 0 \ \forall i.$$

Thus after equating the second sum to zero and substituting for z_i we proceed as in the interpolation case.

These proofs show, therefore, that both the matrices derived from the interpolation and smoothing problems are positive definite, and thus the conjugate gradients method can be applied.

4.3.2. Calculations Involving the Interpolation Matrix.

The implementation of the conjugate gradients algorithm, described above, is straightforward. In both the interpolation and the smoothing cases the preconditioned algorithm requires storage for 4 vectors, of length $2N$ or $3N$ respectively. The algorithm also requires storage for the matrices involved. The amount of storage required depends on the methods used for generating the products Ax and solving the systems of equations $C\gamma=g$. It is these aspects of the algorithm which concern us here. Firstly we consider the interpolation case.

Recall from above that the matrix involved in the interpolation problem satisfies the following,

$$x^{\mathrm{T}}Ax=\sum_{i=1}^{N}\sum_{j\neq i}v_{ij}\left\{\|g_j-g_i\|^2+\frac{\theta_2}{\|t_j-t_i\|^2}[(g_j+g_i)^{\mathrm{T}}(t_j-t_i)]^2\right\}$$

where x is defined as before, to be the *block* vector of all the fitted gradients at the data sites. In what follows let $x_{\underline{k}}$ denote that 2×1 block of the vector x which corresponds to g_k, (that is $(x_{2k-1},x_{2k})^{\mathrm{T}}$). Differentiating the above expression for $x^{\mathrm{T}}Ax$ with respect to the vector g_k we find,

$$(Ax)_{\underline{k}}=\sum_{i=1}^{N}\sum_{j\neq i}v_{ij}\left\{(g_j-g_i)(\delta_{jk}-\delta_{ik})+\frac{\theta_2}{\|t_j-t_i\|^2}[(g_j+g_i)^{\mathrm{T}}(t_j-t_i)(t_j-t_i)(\delta_{jk}+\delta_{ik})]\right\}$$

where δ_{ik} is the Kronecker delta. Summing out the Kronecker deltas we get,

$$(Ax)_{\underline{k}}=2\sum_{i=1}^{N}v_{ik}\left\{(g_k-g_i)+\frac{\theta_2}{\|t_k-t_i\|^2}(g_k+g_i)^{\mathrm{T}}(t_k-t_i)(t_k-t_i)\right\}.$$

This gives, immediately, one way of calculating the product Ax without actually storing the matrix A. The alternative method is to store the matrix in memory and use matrix multiplication to find the product. For this method, and to calculate some of the preconditioning matrices, we need to find expressions for the elements of A. Differentiating $Ax_{\underline{k}}$ with respect to g_l we find that, extending the notation accordingly,

$$A_{\underline{kl}}=2\sum_{i=1}^{N}v_{ik}\left\{(\delta_{kl}-\delta_{il})I_2+\frac{\theta_2}{\|t_k-t_i\|^2}(\delta_{kl}+\delta_{il})(t_k-t_i)(t_k-t_i)^{\mathrm{T}}\right\}.$$

Thus we arrive at the expressions,

$$\tfrac{1}{2}A_{\underline{kl}} = \begin{cases} \dfrac{v_{kl}\theta_2}{\|t_k - t_l\|^2}(t_k - t_l)(t_k - t_l)^{\mathrm{T}} - v_{kl}I_2 & \text{if } l \neq k \\ (\sum_{i=1}^{N} v_{ki})I_2 + \theta_2 \sum_{i=1}^{N} \dfrac{v_{ki}}{\|t_k - t_i\|^2}(t_k - t_i)(t_k - t_i)^{\mathrm{T}} & \text{if } l = k \end{cases}$$

We also require the vector b from $Ax - b = 0$ and using similar techniques to above we see that,

$$x^{\mathrm{T}}b = 4\sum_{i=1}^{N}\sum_{j \neq i} \frac{v_{ji}\theta_2}{\|t_j - t_i\|^2}(g_j + g_i)^{\mathrm{T}}(t_j - t_i)(Z_j - Z_i),$$

therefore,

$$b_{\underline{k}} = 4\sum_{i=1}^{N} \frac{v_{ki}\theta_2}{\|t_k - t_i\|^2}(Z_k - Z_i)(t_k - t_i).$$

Thus we can calculate A and b for use in the conjugate gradients algorithm. For calculating Ax we have two methods. Either we can use the above expression for this product or if the preconditioning requires that we store the parts of the matrix A anyway, we can save a small amount of time by using matrix multiplication, (still remembering that the matrix is sparse).

4.3.3. Calculations Involving the Smoothing Matrix.

In section 4.3.1. we saw that, when A is the smoothing matrix, we have,

$$x^{\mathrm{T}}Ax = 2\sum_{i=1}^{N} z_i^2 + \alpha \sum_{i=1}^{N}\sum_{j \neq i} v_{ij}\left\{\|g_j - g_i\|^2 + \frac{\theta_2}{\|t_j - t_i\|^2}[(g_j + g_i)^{\mathrm{T}}(t_j - t_i) - 2(z_j - z_i)]^2\right\}.$$

In this case x is a $3 \times N$ vector which consists of N 3×1 vectors, which each contain the fitted value and gradients at a data site. Thus in what follows we use $x_{\underline{k}}$ to mean the 3-vector corresponding to the variables at the k-th data site, namely,

$$x_{\underline{k}} = \begin{bmatrix} x_{3k-2} \\ x_{3k-1} \\ x_{3k} \end{bmatrix} \equiv \begin{bmatrix} z_k \\ g_k \end{bmatrix}.$$

The corresponding definitions are taken for $A_{\underline{kl}}$ etc. It is more convenient to replace A in the above with $\alpha^{-1}A$, and we do this.

We proceed in a similar way to the interpolation case, although in this case we differentiate with respect to z_k and g_k in order to find $(Ax)_{\underline{k}}$. Thus we obtain, on differentiating with respect to z_k,

$$2\alpha^{-1}\sum_{i=1}^{N} z_i\delta_{ik} + 2\theta_2\sum_{i=1}^{N}\sum_{j\neq i}\frac{v_{ij}}{\|t_j-t_i\|^2}[(g_j+g_i)^{\mathrm{T}}(t_j-t_i)-2(z_j-z_i)](\delta_{ik}-\delta_{jk})$$

and summing out the Kronecker deltas, as before, we get the first component of $(Ax)_{\underline{k}}$. Differentiating with respect to g_k gives us the other components and we have, in partitioned form,

$$((Ax)_{\underline{k}})_1 = 2\alpha^{-1}z_k + 4\sum_{i=1}^{N}\frac{v_{ik}\theta_2}{\|t_k-t_i\|^2}[2(z_k-z_i)-(g_i+g_k)^{\mathrm{T}}(t_k-t_i)]$$

and

$$((Ax)_{\underline{k}})_{\binom{2}{3}} = 2\sum_{i=1}^{N} v_{ik}\left\{(g_k-g_i)+\frac{\theta_2}{\|t_k-t_i\|^2}[(g_k+g_i)^{\mathrm{T}}(t_k-t_i)-2(z_k-z_i)](t_k-t_i)\right\}.$$

This gives a means for calculating $(Ax)_{\underline{k}}$ when we do not store A, however as before we still need expressions for the entries of the matrix.

We introduce the following notation,

$$X_{kl} = \frac{1}{\|t_k-t_l\|^2}(t_k-t_l)(t_k-t_l)^{\mathrm{T}} \quad \text{and} \quad w_{kl} = \frac{v_{kl}}{\|t_k-t_l\|^2}$$

and continuing with the differentiation, we find,

$$\tfrac{1}{2}A_{\underline{kl}} = \begin{bmatrix} -4\theta_2 w_{kl} & 2\theta_2 w_{kl}(t_k-t_l)^{\mathrm{T}} \\ -2\theta_2 w_{kl}(t_k-t_l) & v_{kl}(\theta_2 X_{kl}-I_2) \end{bmatrix}$$

provided $k\neq l$, and,

$$\tfrac{1}{2}A_{\underline{kk}} = \begin{bmatrix} \alpha^{-1}+4\theta_2\sum_{i=1}^{N} w_{ki} & 2\theta_2\sum_{i=1}^{N} w_{ki}(t_i-t_k)^{\mathrm{T}} \\ 2\theta_2\sum_{i=1}^{N} w_{ki}(t_i-t_k) & \sum_{i=1}^{N} v_{ki}(I_2+\theta_2 X_{ki}) \end{bmatrix}.$$

It only remains to find the appropriate b. We recall that,

$$x^{\mathrm{T}}b = 2\alpha^{-1}\sum_{i=1}^{N} Z_i z_i$$

so that clearly,

$$b_{\underline{k}} = \begin{bmatrix} 2\alpha^{-1}Z_k \\ 0 \\ 0 \end{bmatrix}.$$

Thus we have all the information required to use the conjugate gradients algorithm on the smoothing problem, except that the preconditioning matrices have yet to be

determined.

4.3.4. The Preconditioning Matrices – Interpolation.

In the case of interpolation we have used four preconditioning methods, namely, element-wise diagonal scaling, block diagonal scaling, element-wise SSOR, and block SSOR. We have used two versions of block SSOR, one involving the storage of the matrix A and the other involving recalculation at each iteration. The second of these we expect to be slower, but obviously requires less storage. The recalculation technique seems to have advantages from a numerical point of view, as we shall see later. We have not implemented the incomplete Cholesky conjugate gradients method, mentioned in an earlier section, for the interpolation case, as this method appears to be overly complex for this relatively well conditioned problem. Neither have we looked at the unpreconditioned algorithm in detail, since preliminary investigation indicated that it was far inferior, in performance, to even the simplest preconditioned algorithm.

The simplest case to consider is the element-wise diagonal scaling preconditioning. This method requires only $2N$ storage locations to store the inverses of the diagonal elements of A, provided that the recalculation method for finding Ax is used. The expressions for this product and the diagonal elements are given in section 4.3.2 and involve sums over the neighbours of each data site to find the corresponding diagonal entry in A. Once these entries have been reciprocated, the solution of the preconditioning equations $C\gamma=g$ at each iteration involves simply $2N$ scalar multiplications.

Block diagonal scaling is hardly more complicated. Only the inverses of the 2×2 diagonal blocks of the matrix A need be stored and since these are symmetric they require 3 locations each, so that $3N$ storage locations are required in all. Again the recalculation method of finding Ax is used. Once the diagonal blocks are calculated they are inverted, and then at each iteration N 2 by 2 matrix multiplications are performed. Each matrix multiplication involves 4 scalar multiplications and 2 additions, as opposed to the equivalent 2 scalar multiplications in the element-wise scaling. This, and the fact that inverting N 2 by 2 blocks costs more than $2N$ scalar inversions, means that block scaling has a larger initialisation work load and requires more work per iteration than element-wise scaling. We therefore require a significant reduction in the number of iterations of the algorithm in order to justify the use of this method. Likewise, both the SSOR techniques require more storage (possibly) and more computation both before and at each iteration, so that significant reductions in the number of iterations before convergence will be required before the techniques are justified.

The first SSOR method investigated is the element-wise version. For this technique we need the whole of the matrix A in effect since the diagonal entries and the strict lower triangle are required at each iteration. This can either be stored or recalculated, and in this case we chose to store the matrix in order to decrease the amount of work required at each iteration. We can use the fact that all the blocks involved are symmetric and so require 3 storage locations for each stored block. We know that on average there are 7 non-zero blocks in each (block) row, but since the matrix A itself is symmetric we only need store half (on average) of the off-diagonal blocks. Thus we expect to store $3N$ off diagonal blocks and N diagonal blocks, making $12N$ storage locations in all. In practice this is a slight over-estimate since the data sites near the boundary have generally fewer than 6 neighbours and so there corresponding (block) row in the matrix has fewer non-zero entries. As a spin-off we no longer need to use the recalculation method of finding the product Ax since we are storing A, and we can simply use a sparse matrix multiplication at each iteration.

To solve the preconditioning equations involves the inversion of a $2N \times 2N$ sparse lower triangular matrix, a diagonal matrix and a sparse upper triangular matrix, both of the same size. This costs on average $32N$ scalar multiplications and divisions, and some similar number of additions or subtractions.

The block form SSOR is similar to the element-wise SSOR except that all the operations treat the 2 by 2 blocks as blocks. This means that at each iteration we need the strict block lower triangle and the diagonal blocks and the inverses of the diagonal blocks. If we calculate the inverses of the diagonal blocks at each iteration, this incurs no extra storage over element-wise SSOR, however in the interests of efficiency we decided to store the inverses, so that total storage required is now $15N$ on average. Again the product Ax can be found by sparse matrix multiplication. At each iteration the block equivalents of the above mentioned operations are preformed, resulting in an average of $9N$ 2×2 matrix multiplications, which is equivalent to $36N$ scalar multiplications. There is also an increase in the required number of additions.

During the testing of the above methods it became apparent that the recalculation method for finding Ax at each iteration was hardly slower than the sparse matrix multiplication method. Also the expression used to find the product seemed to be numerically more stable in the cases where the matrix is very ill-conditioned, (which occurs mostly when smoothing). Thus for comparison we have implemented a version of block SSOR preconditioning that only stores the diagonal blocks and their inverses, reducing the required storage to only $6N$ locations.

All the preconditioned algorithms have an overhead involving the calculation and storage of the Dirichlet tessellation and the associated BOD weights. This calculation is carried out once before the iterations start, and the information is stored and kept available throughout the iterations. The solution obtained at the termination of the iterations is the vector of fitted gradients at the data sites, which then can be used in stage two of the natural neighbour spline interpolation process.

4.3.5. The Preconditioning Matrices – Smoothing.

In the case of smoothing we have used the four preconditioning techniques mentioned above, and the method of incomplete Cholesky factorisation. Since, as we shall see later, the SSOR preconditioning that uses no extra storage is considerably slower for large data sets, (precisely when a reduction in storage requirements is valuable), than is the version of SSOR that stores the lower triangle of the matrix being inverted, we have not considered this space saving version in the smoothing case.

The preconditioning techniques of diagonal scaling, SSOR and their associated block derivatives, have analogous implementations in the smoothing case to those in the interpolation case, except that more storage is required. The matrix to be inverted, A, is in this case an $N \times N$ matrix of 3×3 blocks, rather than the 2×2 blocks encountered earlier. The diagonal blocks remain symmetric, as does the matrix A itself, whereas the off-diagonal are no longer so. They have a partly skew symmetric, partly symmetric form as shown below,

$$\begin{bmatrix} a_{11} & -a_{21} & -a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

Economies can still therefore be made, and only the lower triangle of these blocks stored, with the upper triangle being determined by the above pattern. Each block therefore requires 6 storage locations, as opposed to 3 in the interpolation case.

Thus we see that the following average storage requirements apply,

Method	Storage locations
Diagonal Scaling	$3N$
Block Diagonal Scaling	$6N$
SSOR	$24N$
block SSOR	$30N$

The above methods of preconditioning, are very slow to converge, especially when larger numbers of data sites are being considered, in the smoothing case. This suggests that the smoothing matrices are less well conditioned than the interpolation matrices, and this is borne out to some extent by the eigenvalue analysis (see later). For this reasons the more sophisticated incomplete Cholesky factorisation method of preconditioning was used in this case. We shall denote this method by IC.

The implementation of the factorisation requires more storage than any of the other techniques. We require storage for a lower triangular matrix with the superimposed sparsity pattern of the matrix A. Unfortunately we cannot make savings in the storage of the off diagonal blocks because in the factorisation process they lose their special form. Thus we must store the whole of the off diagonal blocks, and the lower triangles of the diagonal blocks. We also require $3N$ locations to store the inverses of the diagonal elements, if they are to be stored, otherwise they can be recalculated at each iteration. We adopt the latter approach. Thus we require 9 storage locations for the off diagonal blocks and 6 for the diagonal blocks. On average this totals $33N$ storage locations for the preconditioning matrix.

Unfortunately the matrix stored in these locations is not the matrix A as in the SSOR cases, so we either have to store this matrix seperately (at a cost of $24N$ more locations) or we can use the recalculation method of finding the products Ax at each iteration. Since the recalculation approach appears numerically more stable in the case of ill-conditioned matrices, we use it in our implementation.

The closeness of the incomplete factorisation to the full factorisation depends upon a good ordering for the rows and columns of A. The closer all of the non-zero entries are to the diagonal the better the approximation. For our purposes simply sorting the data sites by their projections onto a straight line is sufficient, and this is accomplished using a simple insertion sort. All the other preconditioning methods do not sort the data sites in our implementations, so in the results presented later the timings for IC include the sort time. This time could clearly be reduced by the use of a more complicated sorting algorithm.

Let us now consider the factorisation process. The algorithm for the factorisation, taking account of the enforced sparsity becomes,

$$
\begin{aligned}
&\text{for } i = 1,\ldots,3N \\
&\quad D_{ii} = A_{ii} - \sum_k L_{ik}^2 D_{kk} \\
&\quad \text{for } j > i \text{ such that } A_{ji} \neq 0 \\
&\qquad L_{ji} = (A_{ji} - \sum_k L_{jk} L_{ik} D_{kk}) D_{ii}^{-1}
\end{aligned}
$$

where the sums are taken over all $k < i$ with, A_{ik} non-zero in the first case, A_{ik} and A_{jk} both non-zero in the second. Here we have modified the algorithm slightly from the version mentioned previously, so that a unit lower triangular matrix and a diagonal matrix are produced. This means that less storage is required, but more work is required to obtain the factorisation. However, using this form, less work is required at each iteration to invert the preconditioning matrix.

The conditions, $A_{ji} \neq 0$ etc., are equivalent to the condition that the jth and ith variables be associated with data sites that are either the same or neighbours. Thus for $i \equiv j$ the condition that A_{ik} be non-zero is simply equivalent to i and k being associated with the same or neighbouring data sites, and for $i \neq j$ the conditions are equivalent to the data sites associated with the three variables concerned being pairwise either the same or neighbours. This essentially the way in which the factorisation is implemented, except that the summations are accrued in a different order and the diagonal entries are first modified by the factor $(1+\eta)$, for some suitable η. The various matrices are stored in the, previously discussed, block form since this is convenient.

It must be remembered that the above version of the algorithm is not applicable as a block-wise factorisation. It can however be modified, subject to the proviso that no diagonal block becomes non-positive definite during the factorisation. The modified version has the following form,

$$
\begin{aligned}
&\text{for } i = 1,\ldots,N \\
&\quad D_{ii} = A_{ii} - \sum_k L_{ik} D_{kk} L_{ik}^{\mathrm{T}} \\
&\quad \text{for } j > i \text{ such that } A_{ji} \neq 0 \\
&\qquad L_{ji} = (A_{ji} - \sum_k L_{jk} D_{kk} L_{ik}^{\mathrm{T}}) D_{ii}^{-1}
\end{aligned}
$$

We have not implemented this version, because of the problem of the diagonal blocks D_{ii} becoming non-positive definite.

We have now described the techniques used for inverting the matrices involved in the natural neighbour spline problem, and discussed their implementation. In the next section we present the results of some tests carried out to try to evaluate the performance of these techniques, when applied to various problems.

4.4. Some Examples of the use of Preconditioned Conjugate Gradients Method for Natural Neighbour Splines.

4.4.1. The Interpolation Case.

In order to compare the efficiency of the implementations of the preconditioning techniques described in the previous sections, we have used four sets of data sites. The four sets consist of 10, 100, 1000 and 10000 points randomly scattered in the square $0 \leq x,y \leq 1$. We have also interpolated four different functions at each of these sets of sites, but the resulting number of iterations required are so similar, that we report only the results of one interpolation in each case here. Thus we have four interpolation problems with widely differing numbers of data sites. Recall that the thin plate spline procedure would not be able to interpolate to the 1000 or 10000 point data sets, with the current full matrix inversion methods that we use, and the execution times for a 100 point data set can be found in chapter 2.

We have used five preconditioning techniques in the solution of the interpolation problems. These are element-wise diagonal scaling, block diagonal scaling, element-wise and block SSOR, and an implementation of block SSOR that requires no more storage than block diagonal scaling. We have not used incomplete Cholesky decomposition for the interpolation problem, since as the results show, it is unnecessary to use such a complicated procedure for this case given that the simpler methods perform so well.

Before we look at the execution times for the various methods, we discuss briefly the condition number of the unpreconditioned matrix. In order that we may use existing full matrix routines to calculate the eigenvalues required we look only at the 10 point case. In this case we must find the largest and smallest eigenvalues of a 20×20 matrix. The resulting eigenvalues, for the 10 point problem also considered later, are 1.74 and 240.38, which leads to a condition number of about 140. This is not excessively large, and so this interpolation problem is quite well conditioned. If we precondition this matrix, using the simplest preconditioning (element-wise diagonal scaling), this is equivalent to pre- and post-multiplying the matrix by the diagonal matrix whose entries are the inverses of the square roots of the diagonal entries of the unpreconditioned matrix. The eigenvalues of this preconditioned matrix are 0.0482 and 2.33, which leads to a condition number of about 50. So even this simple form of conditioning can reduce the condition number by a factor of almost 3.

A final point must be made about the convergence criterion. Recall from the description of the conjugate gradients algorithm, that we iterate until $r_{i+1}^{\mathrm{T}}\gamma_{i+1}$ is less than some specified value, where r_{i+1} is the residual vector $Ax-b$ at the $i+1$th step, and $\gamma_{i+1}=C^{-1}r_{i+1}$, with C the preconditioning matrix. This quantity is calculated by the algorithm for use elsewhere, so checking for convergence incurs no extra computation when using this criterion. However it is easy to see that the criterion depends upon the preconditioning being used, so that in order to compare preconditioning techniques we should use some criterion which does not have this dependence. The obvious choice is the sum of squared residuals, namely $r_{i+1}^{\mathrm{T}}r_{i+1}$, which means extra computation at each iteration. This computation is relatively small, however, and is constant between preconditioning methods. In our programs the algorithm was said to have converged when $r_{i+1}^{\mathrm{T}}r_{i+1} < 10^{-15}$.

As an initial approximation for all the methods we took that which corresponds to zero gradient at all the data sites. In fact the initial approximation made little difference to the number of iterations required for convergence. All the routines are implemented in single precision, in FORTRAN 77, with double precision accumulation of inner products.

In the tables which follow we list the number of iterations to convergence, the CPU time required on a SUN-4/260 workstation using optimised code, and the final value of $r_{i+1}^{\mathrm{T}}\gamma_{i+1}$, for each preconditioning method used on each of the four interpolation problems.

10 point problem.

Preconditioner		No. of iterations	CPU Time (secs)	$r_{i+1}^{\mathrm{T}}\gamma_{i+1}$ $\times 10^{-15}$
Diagonal Scaling	(element-wise)	22	<0.1	0.0004
	(block)	19	<0.1	0.05
SSOR	(element-wise)	13	<0.1	0.004
	(block)	12	<0.1	0.000001
	(block reduced storage)	12	<0.1	0.000001

100 point problem.

Preconditioner		No. of iterations	CPU Time (secs)	$r_{i+1}^{T}\gamma_{i+1}$ $\times 10^{-15}$
Diagonal Scaling	(element-wise)	37	0.4	0.024
	(block)	25	0.3	0.015
SSOR	(element-wise)	16	0.3	0.006
	(block)	12	0.3	0.022
	(block reduced storage)	12	0.4	0.022

1000 point problem.

Preconditioner		No. of iterations	CPU Time (secs)	$r_{i+1}^{T}\gamma_{i+1}$ $\times 10^{-15}$
Diagonal Scaling	(element-wise)	71	8.4	0.016
	(block)	41	5.6	0.044
SSOR	(element-wise)	29	6.4	0.004
	(block)	21	5.1	0.005
	(block reduced storage)	21	7.9	0.005

10000 point problem.

Preconditioner		No. of iterations	CPU Time (secs)	$r_{i+1}^{T}\gamma_{i+1}$ $\times 10^{-15}$
Diagonal Scaling	(element-wise)	84	109.9	0.023
	(block)	48	73.0	0.022
SSOR	(element-wise)	34	85.8	0.012
	(block)	24	65.3	0.022
	(block reduced storage)	24	96.3	0.022

Note that in the 10 point problem the execution times are faster than the resolution of the timing mechanism used.

In all cases the block techniques take fewer iterations than the element-wise techniques, in fact sufficiently fewer that the CPU times are significantly reduced despite the extra complexity of each iteration. Also in all but the 10 point case the block SSOR method requires about half the number of iterations that the block diagonal scaling does. The reduction in execution time however is about 10% in the version that uses extra storage, whereas the version of block SSOR that uses no more storage than block diagonal scaling is actually slower because of the increased complexity of each iteration. It can be seen therefore that much of the reduction in CPU time gained by the use of the block SSOR method, requires that the whole of the matrix being inverted be stored. So it would appear that in the case of interpolation, block diagonal scaling preconditioned conjugate gradients is the most economical technique (in time and storage) of those we have

investigated. In fact only 48 iterations and 1¼ minutes of CPU time were required to reach the desired accuracy for a very large problem indeed. Notice also that $r_{i+1}^{\mathrm{T}}\gamma_{i+1}$ is always smaller than $r_{i+1}^{\mathrm{T}}r_{i+1}$ in our examples, a fact which must be borne in mind when using $r_{i+1}^{\mathrm{T}}\gamma_{i+1}$ to test for convergence.

4.4.2. The Smoothing Case.

In the case of smoothing splines we have used five preconditioning techniques. These are the element-wise and block versions of diagonal scaling and SSOR and the element-wise incomplete Cholesky (IC). We have applied the techniques to the same problems as before, with the addition of the choice of smoothing parameter, $50/N$ where N is the number of data sites. In the IC method we have taken $\eta=(10N)^{-1}$. Again let us first look at the conditioning of the 10 point problem.

The unpreconditioned matrix, which is now 30×30 as opposed to 20×20 in the interpolation case, has smallest and largest eigenvalues which result in a condition number of over 10^6. This makes the smoothing problem quite ill-conditioned, and we expect that much more complicated preconditioning techniques will be required to obtain convergence, particularly in the larger problems. The element-wise diagonally scaled matrix has a condition number which is smaller by a factor of over 20 but still this leaves the matrix very ill-conditioned. Remember that these matrices are only 30×30 so that worse conditioning can be expected from the larger problems.

Investigations also show that the unpreconditioned matrix becomes more ill-conditioned, as the degree of smoothing is increased. This is in one sense counter intuitive, in that as we increase the smoothing the solution surface approaches the least squares fitted first degree function, a very simple surface which is easy to calculate. The reason for the increase in ill-conditioning, however, can be traced to the roughness penalty in the minimisation functional. Recall that if A is the unpreconditioned matrix to be inverted and x a vector consisting of the fitted value and gradient information at the data sites, z_i and g_i then we have,

$$x^tAx=\sum_{i=1}^{N}z_i^2+\alpha\sum_{i=1}^{N}\sum_{j\neq i}v_{ij}\left\{\|g_j-g_i\|^2+\frac{\theta_2}{\|t_j-t_i\|^2}[(g_j+g_i)^{\mathrm{T}}(t_j-t_i)-2(z_j-z_i)]^2\right\}.$$

Setting the roughness contribution to zero, we see that

$$\sum_{i=1}^{N}\sum_{j\neq i}v_{ij}\|g_j-g_i\|^2=0 \;\Rightarrow\; g_j=g_i=g \text{ (say) } \forall\, i,j$$

and the second term in the roughness component then gives,

$$g^{\mathrm{T}}(t_j-t_i)-(z_j-z_i)=0 \text{ whenever } i \text{ and } j \text{ are neighbours.}$$

This implies that $z_i-g^{\mathrm{T}}t_i$ is a constant for all i. So we have shown that the roughness contribution to $x^{\mathrm{T}}Ax$ vanishes if and only if x corresponds to the values and gradients on a plane, and we are left solely with $\sum z_i^2$. So we can think of A as being in the form $E+\alpha R$ where E is a matrix with the repeating pattern (1,0,0) along the diagonal and zeros elsewhere and R is such that $x^{\mathrm{T}}Rx$ is the roughness contribution. We can see that E has $2N$ zero eigenvalues, and R, as shown above, has only 3 zero eigenvalues (since the first degree functions are 3 parameter family), but the corresponding eigenspaces have zero intersection since, as we have seen, $E+\alpha R$ is positive definite for all values of $\alpha>0$. As α increases however, the matrix E becomes less important in the sum and $(E+\alpha R)x\approx\alpha Rx$ so that A will have 3 very small eigenvalues.

Axelsson (1976) has shown that the number of iterations, k, required to reach a given relative accuracy (measured by,

$$\left[\frac{(x_k-\hat{x})^{\mathrm{T}}A(x_k-\hat{x})}{(x_0-\hat{x})^{\mathrm{T}}A(x_0-\hat{x})}\right]^{\frac{1}{2}}$$

where $\hat{x}$ is the solution, x_0 is the initial approximation and x_k is the result of the kth iteration), is directly proportional to the square root of the spectral condition number of A ($C^{-\frac{1}{2}}AC^{-\frac{1}{2}\mathrm{T}}$ in the preconditioned case). Thus as the smoothing increases the required number of iterations will increase since the condition of $E+\alpha R$ tends to the condition number of αR and R has an infinite condition number.

Fortunately we know the solution when $\alpha=\infty$, namely the least squares fitted first degree function to the data. In practice, therefore, when smoothing heavily we can reduce the number of iterations required by starting from this zero roughness solution. On the other hand when smoothing lightly, fewer iterations result when we use an initial approximation closer to the interpolant. There is an area in the centre of the smoothing range however when some of the preconditioning techniques perform quite badly, and this aspect requires further investigation. Later we show the effect of changing the smoothing parameter on one example, but first we present the results of the comparison of preconditioning techniques. $50/N$ was chosen as the smoothing parameter for these problems, and it appears from observation of the solution surface, that this is slightly over-smoothing. The starting point used in all cases, was that corresponding to values matching the data and zero gradient at all the data sites. Convergence was checked for as before using $r_{i+1}^{\mathrm{T}}r_{i+1}$, and the algorithm terminated after 1000 iterations if convergence

had not been reached by then. The results are tabulated below.

10 point problem.

Preconditioner		No. of iterations	CPU Time (secs)	$\log_{10} r_{i+1}^T r_{i+1}$	$\log_{10} r_{i+1}^T \gamma_{i+1}$
Diagonal Scaling	(element-wise)	166	0.1	-15	-20
	(block)	101	0.1	-15	-18
SSOR	(element-wise)	525	1.2	-15	-18
	(block)	203	0.4	-15	-18
IC	(element-wise)	28	<0.1	-17	-20

100 point problem.

Preconditioner		No. of iterations	CPU Time (secs)	$\log_{10} r_{i+1}^T r_{i+1}$	$\log_{10} r_{i+1}^T \gamma_{i+1}$
Diagonal Scaling	(element-wise)	1000	13.4	-9	-13
	(block)	830	11.8	-15	-19
SSOR	(element-wise)	1000	26.7	-9	-13
	(block)	621	16.1	-15	-19
IC	(element-wise)	71	1.7	-15	-18

1000 point problem.

Preconditioner		No. of iterations	CPU Time (secs)	$\log_{10} r_{i+1}^T r_{i+1}$	$\log_{10} r_{i+1}^T \gamma_{i+1}$
Diagonal Scaling	(element-wise)	1000	133.8	3	-4
	(block)	1000	141.2	2	-4
SSOR	(element-wise)	1000	286.9	5	-3
	(block)	1000	286.6	3	-4
IC	(element-wise)	275	69.5	-15	-19

10000 point problem.

Preconditioner		No. of iterations	CPU Time (secs)	$\log_{10} r_{i+1}^T r_{i+1}$	$\log_{10} r_{i+1}^T \gamma_{i+1}$
Diagonal Scaling	(element-wise)	1000	1424.0	3	-4
	(block)	1000	1504.2	2	-5
SSOR	(element-wise)	1000	3100.8	4	-3
	(block)	1000	3093.1	1	-6
IC	(element-wise)	283	851.6	-15	-20

Notice here that, except possibly for the smallest data set, the IC preconditioning consistently out performs all the other methods both in numbers of iterations

required and in execution times. In fact this method converges in less than 300 iterations in all the examples. The decrease in $r_i^T r_i$ as the algorithm proceeds is quite erratic, and by no means monotonic, so the final values can vary by about a factor of 10. This erratic behaviour is due to the fact that the conjugate gradients algorithm is designed to minimise the functional, x^TAx-2b^Tx not the sum of squared residuals, $(Ax-b)^T(Ax-b)$. In the context of preconditioning $r_{i+1}^T\gamma_{i+1}$ corresponds to $x^TA(C^{-1}A)x-2b^T(C^{-1}A)x$, so that if C is a good preconditioner and also has the property that $C^{-1}A\approx I$, then $r_{i+1}^T\gamma_{i+1}$ is approximately the functional we are minimising.

We now return to the problem of ill-conditioning which increases with the smoothing parameter. To investigate this more fully we have taken the 100 point problem and used IC preconditioned conjugate gradients until convergence, for 7 different values of the smoothing parameter. (0.005, 0.05, 0.5, 5.0, 50.0, 500.0, and 5000.0). These values have a range corresponding to a wide range of smoothing. In the table below we list the corresponding number of iterations when using two different starting values, namely, (1) the least squares fitted plane, and (2) the values matching the data and zero gradients starting values mentioned earlier.

α	No. of iterations	
	start (1)	start (2)
0.005	28	28
0.05	49	51
0.5	69	71
5.0	98	102
50.0	95	101
500.0	101	128
5000.0	125	142

At small values of α the number of required iterations are very similar, and this is consistent with our observations in the interpolation case where the choice of starting value made little difference. In the case of smoothing with some other preconditioners the start (2) performed better than start (1) for these lower values of the smoothing parameter. When the smoothing increases, start (1) (the smooth start) does better than start (2) by up to about 20%, and when using some other preconditioners can make the difference between convergence and non-convergence (in less than 1000 iterations).

Overall we conclude that in the case of smoothing the use of the complicated preconditioner (incomplete Cholesky) is worthwhile in terms of efficiency, whenever the extra storage is available. For small problems block diagonal scaling is not significantly slower, in execution time, but takes many more

iterations and in these small problems storage is unlikely to be a problem. In the larger problems, the savings of IC preconditioning are so great, and the convergence of the methods (at least in SUN single precision) so unreliable that if the memory demands of IC preconditioning are too great, then finding the solution will be very difficult.

The increased use of double precision in the implementation (possibly with extended precision accumulation of inner products) is one possible way of reducing the effect of some of the problems of numerical instability. In the next chapter we have made brief comparisons of single and double precision routines. However it must be remembered that the use of double precision implies the use of more memory and slower floating point calculations, so that it may not be possible to use the methods on the larger problems and the double precision implementation may be slower.

Chapter 5.
More on Natural Neighbour Splines.
5.1. A Direct Method for Solving the Linear System in the Smoothing Case.

For the case of natural neighbour smoothing splines we have looked briefly at a direct method of solving the system of equations. Recall that the matrix A in the system $Ax=b$ is symmetric and positive definite, so that we can use Cholesky decomposition which is known to be stable.

A good ordering of the data sites (block rows and columns of A) is necessary however, in order to minimise the amount of storage required. This can be understood by observing that during the Cholesky factorisation all of the fill-ins occur in the envelope of A (that part of the matrix from the first non-zero in each row to the diagonal). Thus an ordering which reduces the size of the envelope, reduces the amount of storage required for the factored (and filled in) matrix. In our case, since the sparsity pattern of A is determined by the neighbour relation, and this in turn is determined by the positions of the data sites, a relatively good ordering can be obtained by sorting the data sites according to their projections onto a given line through the points (the first principal component for example). This is the same ordering used in the incomplete Cholesky precondition conjugate gradients method mentioned in the previous chapter.

With this ordering, points close to each other will be close in the ordering, and we expect the number of blocks between the first non-zero block in each block-row of A and the diagonal to be $O(\sqrt{N})$. Thus the overall storage required is approximately $O(N^{3/2})$, compared to $O(N)$ in the iterative methods discussed earlier.

In the following examples, to illustrate the efficiency and accuracy of the direct method, we have used two implementations of the Cholesky factorisation on smoothing problems consisting of data at 10, 100 and 1000 points scattered in the square $\{ 0 \le x, y \le 1 \}$. The implementations differ only in that the first is a single precision implementation (with double precision accumulation of inner products) and the second is a double precision implementation of the same algorithm. (The NAG library additional precision accumulation of inner products was used in the second case.) Single and double precision implementations of the the incomplete Cholesky preconditioned conjugate gradients are also included in the examples for comparison.

Since the direct method provides no guide as to the accuracy of the solution, as part of the algorithm, whereas the iterative methods provide such a measure in the form of the convergence criterion, we have looked at the norm of the difference between the solution given by the double precision iterative technique and the solutions given by the other methods. Also reported are the CPU times for each program on a SUN4/260.

Table of the Norms of the difference between the solutions given by IC conjugate gradients (double precision) and other methods.

		Norms of difference 10 pts	100 pts	1000 pts
Iterative Method	(single precision)	0.0025	0.0002	0.0062
Direct Method	(double precision)	10^{-6}	10^{-6}	10^{-5}
	(single precision)	0.15	0.01	6.91

Table of Execution Times on SUN4/260.

		Run Times (secs) 10 pts	100 pts	1000 pts
Iterative Method	(double precision)	0.1	3.1	110.4
	(single precision)	<0.1	1.5	52.7
Direct Method	(double precision)	<0.1	1.0	112.6
	(single precision)	<0.1	0.9	98.5

The table of norms shows that the single precision iterative method does consistently better than the single precision direct method, using the current convergence criteria. The double precision direct method produces results which are very similar to the double precision iterative method, but of course because they use almost double the storage of the single precision techniques, they are restricted to the smaller problems. In fact the double precision direct method cannot be run on a 10,000 point example on our 32Mbyte SUN workstation, because this is insufficient memory to store the matrix factors. We could of course use backing store for the factors but this would slow the method down quite significantly.

The execution times show that the single precision iterative method competes very well with the direct method, especially for the larger data set. Given the accuracy of the iterative method, it is obviously to be preferred, in most cases, over the direct method considered above. The use of more sophisticated direct methods is

not considered here.

5.2. A Comparison of Natural Neighbour and Finite Window Splines.

In this section we briefly compare the natural neighbour and finite window splines, fitted to three data sets. The first two of these data sets are simulated from relatively simple functions used in earlier chapters. The splines compared are interpolatory. The third data set is one of the real data sets looked at in chapter 3, and in this example we also consider smoothing splines. The natural neighbour splines are contoured using CONICON3, as are the finite window splines. This package requires values and gradients at a grid of points, and a bridging function is used to evaluate the natural neighbour spline away from the data sites. The value and gradient of this bridging function (and therefore the natural neighbour spline) at an arbitrary point inside the window, is determined from the fitted values and gradients at the data sites by minimising a roughness penalty which has the same form as that used to determine the fitted values and gradients at the data sites. The penalty used to determine the value z and gradient g at the point t is,

$$\sum_j \kappa_j(t)\{\|g-g_j\|^2 + \frac{\theta_2}{\|t-t_j\|^2}[(g+g_j)^T(t-t_j)-2(z-z_j)]^2\}$$

where here the weights $\kappa_j(t)$ are the subtile weights at the points t. These subtile weights are found by inserting the point t into the tessellation of the data sites (used to find the BOD weights) and then assigning to $\kappa_j(t)$ the area of that part of the tile associated with t which used to be associated with t_j. That is,

$$\kappa_j(t) = \sigma_2(T_j(t))$$

where

$$T_j(t) = \{\, s\in\Omega \text{ (the window)} \;:\; \|s-t\|<\|s-t_j\|<\|s-t_i\| \;\; \forall\; i\neq j \,\}.$$

These weights are zero whenever t is not a neighbour of t_j, and minimising the penalty produces a z and g that are a weighted averages of the fitted values and gradients at the data sites neighbouring t. Sibson (1985) proves that the bridging function defined in this way is continuously differentiable, a property that would not hold if the BOD weights were substituted for the subtile weights.

It can also be seen that the value and gradient of the natural neighbour spline can only be calculated at points that lie inside the window. Since CONICON3 requires the values and gradients at all four corners of a grid cell before it can contour that cell, the maps of the natural neighbour splines do not go quite up to

the edge of the window. The expression for the value (and gradient) of the finite window splines, however, can be evaluated at any point in the plane, so that the maps of these splines cover a region slightly larger than the window, but are only drawn inside the window.

For more information on the subtile weights and the bridging function see Sibson (1985).

5.2.1 Interpolation.

The two examples used in comparing the interpolatory splines are the functions $4xy$ and $3(x-0.1)^4-2y^2$ observed at 25 randomly scattered points in the square $\{\frac{1}{2} \le x, y \le \frac{1}{2}\}$. In the first example, since the discrete roughness penalty in the natural neighbour approach recovers the roughness of a quadratic exactly (apart from edge effects), we expect both spline interpolants to be very similar. In the case of the finite window spline, 23 extra functions were used, and in both cases the window over which the penalties are calculated was the square in which the data are scattered. Figures 5.2.1 and 5.2.2 show the natural neighbour and finite window spline interpolants respectively. In fact, because of the problem that the natural neighbour spline cannot be evaluated on or outside the window boundary, these contour maps cover a region very slightly smaller than the window. These figures are very similar which is as expected given the above.

The second example function is a quartic and so in this case the discrete penalty is truly an approximation to the roughness. We might expect greater differences in the two fitted splines, but in practice the contour maps are almost identical. They differ only very slightly around the edge of the square of interest. It must be remembered that both approaches are approximations to the true finite window spline. Both the above examples seem to indicate that, at least in the interpolation case and for these simple functions, the distinct approaches result in similar solution splines. The next section considers interpolation to a more complicated function, and also looks at smoothing.

5.2.2 A Real Data set.

The data set used in this example is the first data set of those used in chapter 3. It consists of observations at 38 sites as shown in chapter 3. The window used is that shown in the contour maps that follow. We have fitted the interpolating and two smoothing natural neighbour splines to the data, and for comparison the finite window spline equivalents. In the smoothing case, because of the errors involved in the discrete version of the roughness penalty, the smoothing parameters do not play the same role. However since the error penalty term in both cases is of the

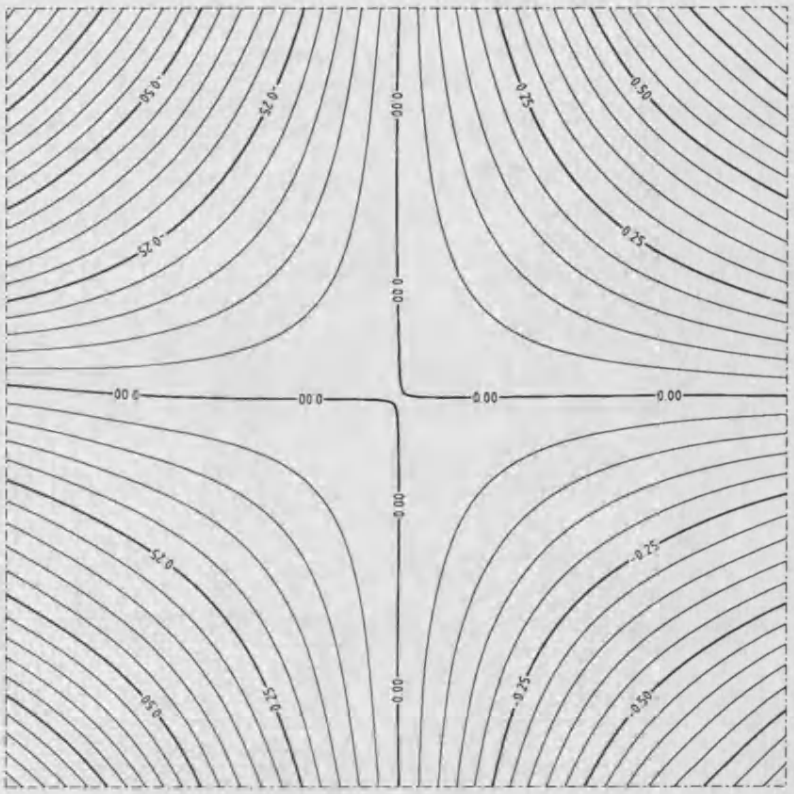


fig 5.2.1 Natural neighbour interpolant

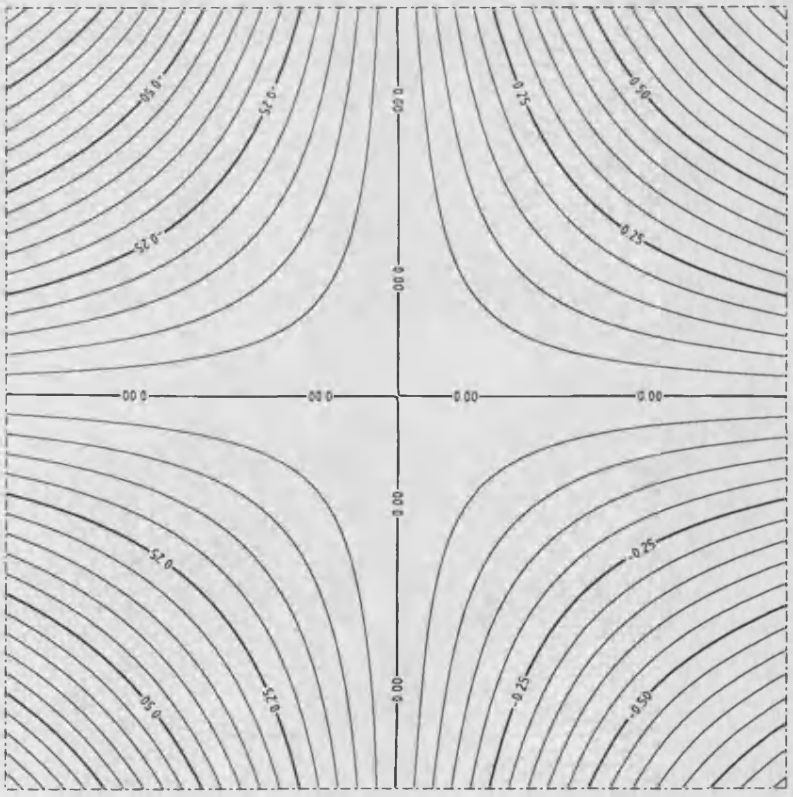


fig 5.2.2 Finite window interpolant

same form, we can use this as guide to the amount of smoothing. Thus we have compared both natural neighbour smoothing splines with the finite window smoothing spline that produces the same residual error.

In the maps which follow notice that the natural neighbour spline cannot be contoured close up to the edge of the window, where the window boundary does not run parallel to the grid directions used for contouring. Figures 5.2.3 and 5.2.4 show the natural neighbour and finite window interpolating splines respectively. Much of the difference occurs around the edge of the window, and this is where the discrete roughness penalty is most inaccurate since the points on the boundary do not have a full set of neighbours. We attribute these differences to this edge effect and some edge correction procedure may be appropriate to improve performance in these areas. The other areas of difference occur where the fitted surface is relatively flat, and these are areas in which contouring is known to present small differences in the height of the surface as large differences in the positions of the contours.

Figure 5.2.5 shows the natural neighbour smoothing spline with smoothing parameter 1.0. This spline has a residual error penalty of 17.34 and the finite window spline with a similar error occurs when a smoothing parameter of 1.44 is used in this case. Figure 5.2.6 shows the finite window spline with this value of the smoothing parameter. These maps have a very similar appearance, except for some edge effects. The splines are closer in appearance than the interpolating splines and this is to be expected since we have reduced the variability in both the fitted surfaces. Figures 5.2.7 and 5.2.8 show the natural neighbour and finite window smoothing splines with values of the smoothing parameter of 10.0 and 11.34 respectively. The residual error in this case is 99.52. The differences in these maps are minor, and seem to occur in the upper right hand quarter of the plot.

The results of this limited study outlined above seem to indicate that natural neighbour splines produce very similar results to the finite window splines discussed earlier in this thesis. This is especially true when a degree of smoothing is applied, or the recovered surface is smooth. When the recovered surface is less smooth, as it is in the case of interpolation to the real data mentioned above, the differences are greater but largely confined to the edge of the data. Clearly given larger data sets the interpolating splines must perform very similarly well inside the data, since the observations will determine the shape of the surface more than the smoothness criteria.

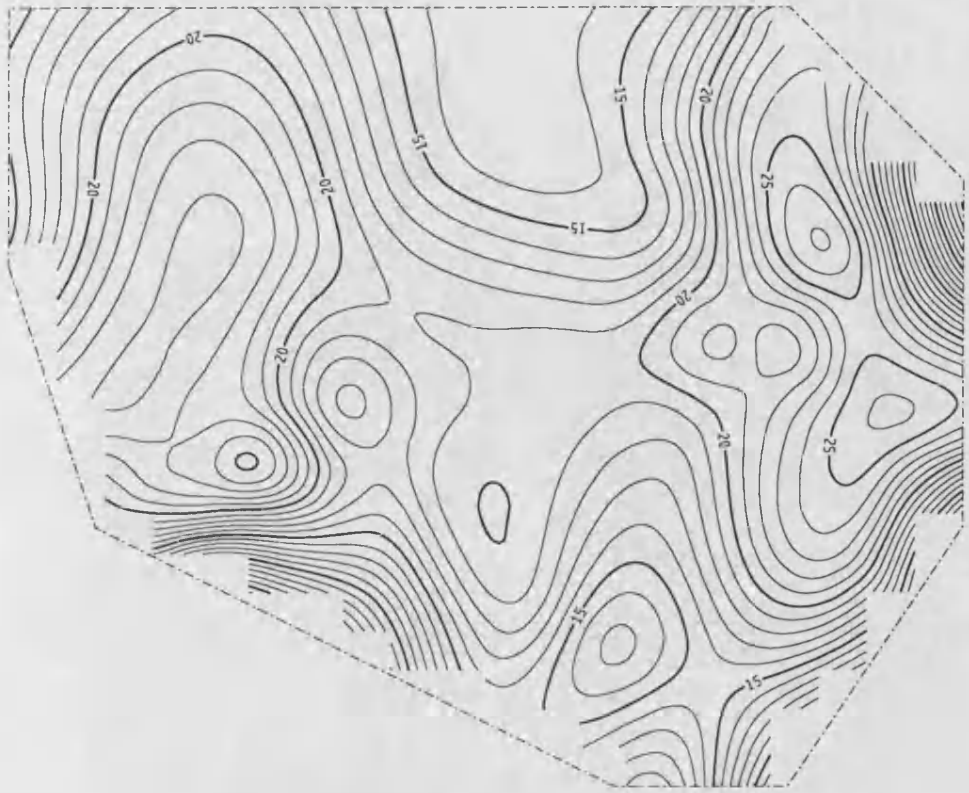


fig 5.2.3 Natural neighbour interpolant

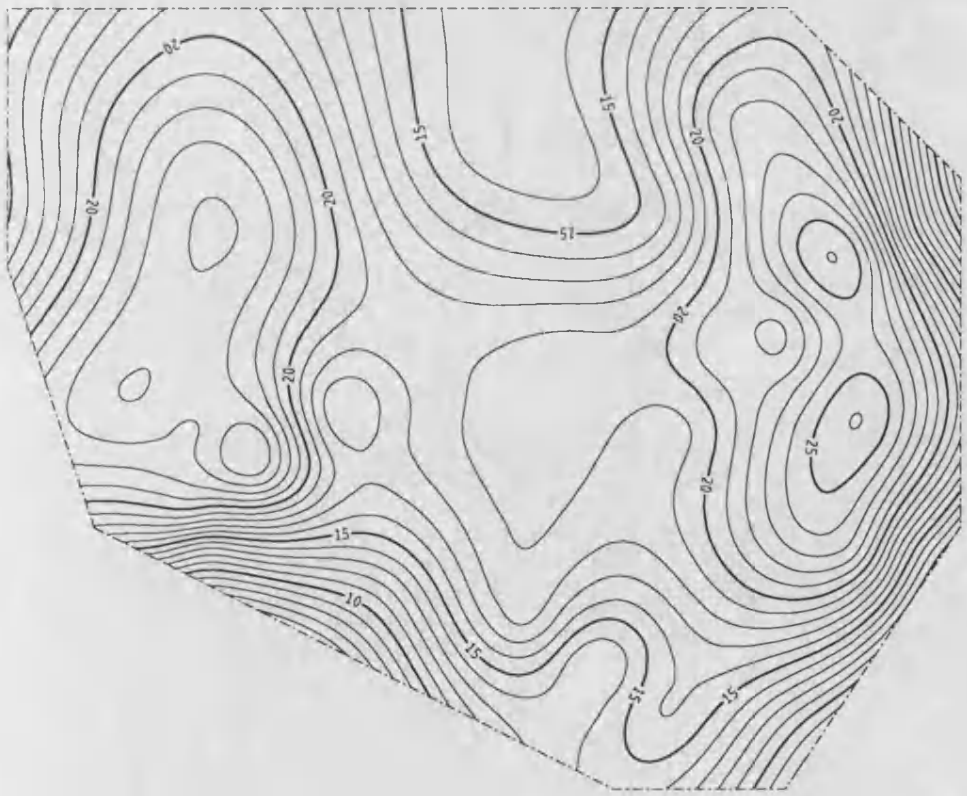


fig 5.2.4 Finite window interpolant

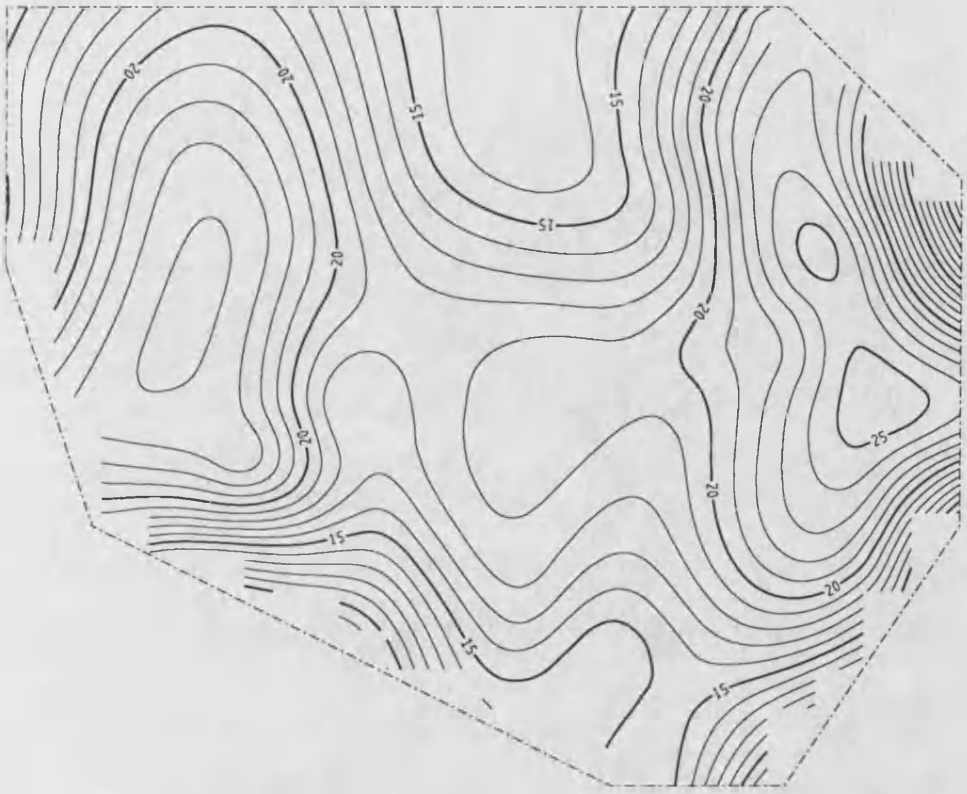


fig 5.2.5 Natural neighbour smoother (alpha=1.0)

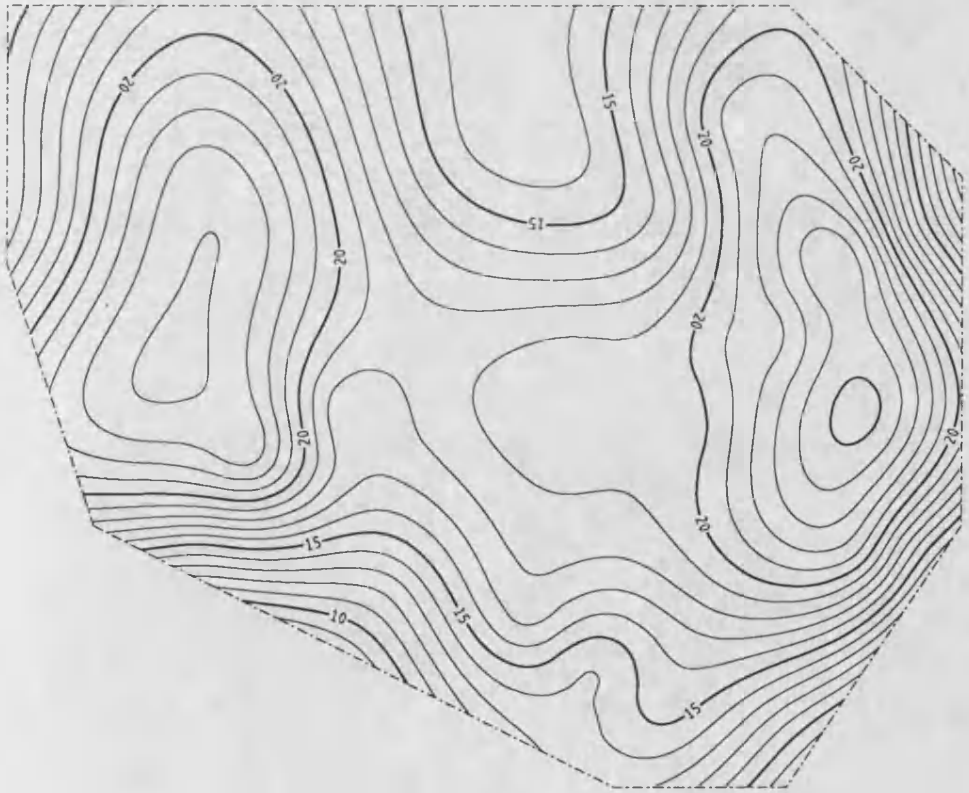


fig 5.2.6 Finite window smoother (error matched)

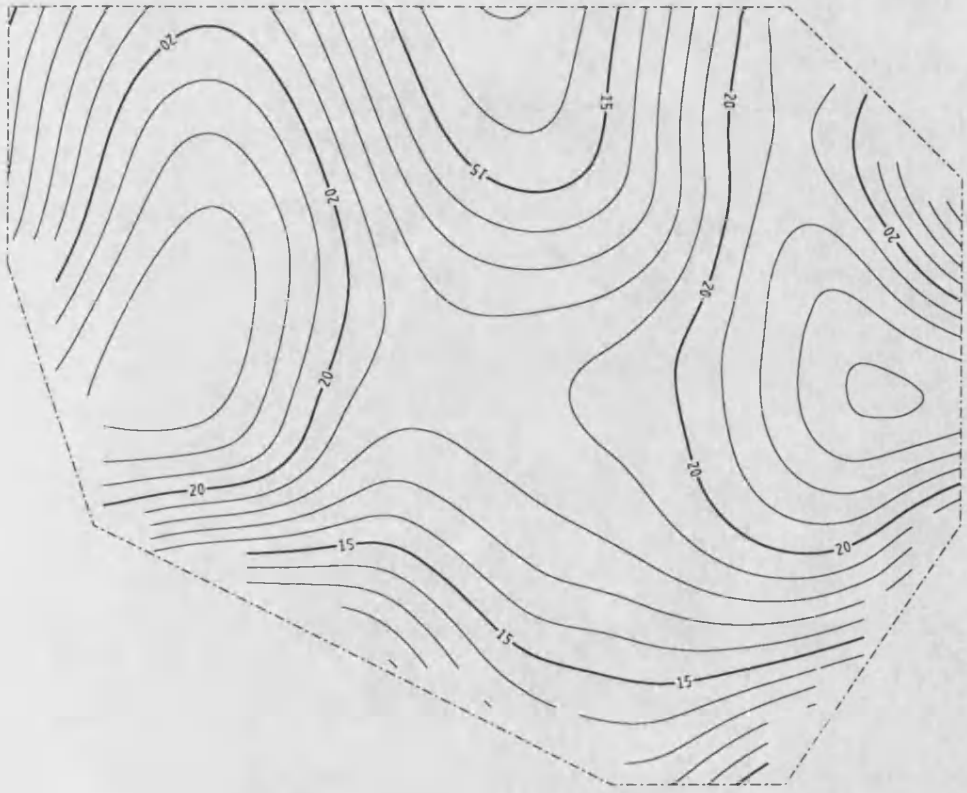


fig 5.2.7 Natural neighbour smoother (alpha=10.0)

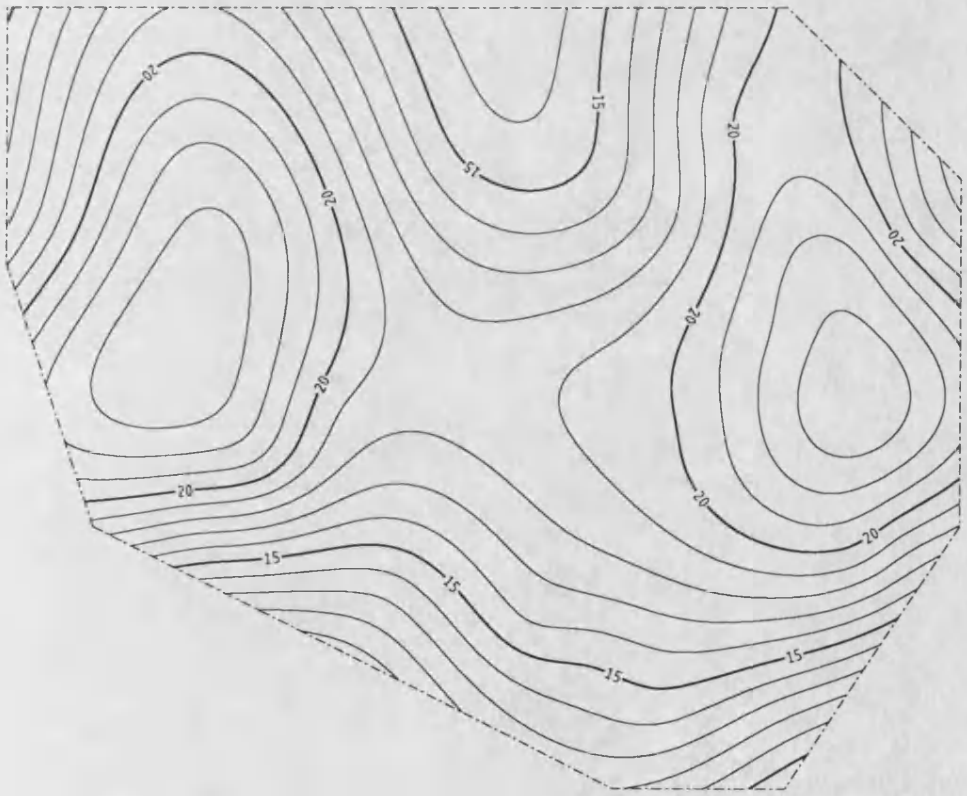


fig 5.2.8 Finite window smoother (error matched)

Thus natural neighbour splines are a very valuable tool for surface fitting, especially for moderately large data sets (more than 200 points). Work has still to be done in the area of automatic choice of smoothing parameter, and in particular the efficient calculation of smoothing splines for a set of different smoothing parameters.

Appendix A.
A Green's Formula for the Roughness Penalty.

In this appendix we present a derivation of a Green's formula for the roughness penalty. A more general derivation can be found in Aubin (1972). The derivation uses the divergence theorem in the plane for generalised derivatives.

We have,

$$A_\Omega(u,v) = \int_\Omega \partial_{xx}u\partial_{xx}v + 2\partial_{xy}u\partial_{xy}v + \partial_{yy}u\partial_{yy}v$$

Firstly we notice that,

$$\begin{aligned} \partial_{xx}u\partial_{xx}v &= \operatorname{div}\begin{bmatrix}\partial_{xx}u\partial_x v\\ 0\end{bmatrix} - \partial_{xxx}u\partial_x v \\ &= \operatorname{div}\begin{bmatrix}\partial_{xx}u\partial_x v - \partial_{xxx}uv\\ 0\end{bmatrix} + \partial_{xxxx}uv \end{aligned} \tag{A.1}$$

And simlarly,

$$\partial_{yy}u\partial_{yy}v = \operatorname{div}\begin{bmatrix}0\\ \partial_{yy}u\partial_y v - \partial_{yyy}uv\end{bmatrix} + \partial_{yyyy}uv \tag{A.2}$$

Also we have that,

$$\begin{aligned} \partial_{xy}u\partial_{xy}v &= \operatorname{div}\begin{bmatrix}\partial_{xy}u\partial_y v\\ 0\end{bmatrix} - \partial_{xxy}u\partial_y v \\ &= \operatorname{div}\begin{bmatrix}\partial_{xy}u\partial_y v\\ -\partial_{xxy}uv\end{bmatrix} + \partial_{xxyy}uv \end{aligned} \tag{A.3}$$

And also

$$\begin{aligned} \partial_{xy}u\partial_{xy}v &= \operatorname{div}\begin{bmatrix}0\\ \partial_{xy}u\partial_x v\end{bmatrix} - \partial_{xyy}u\partial_x v \\ &= \operatorname{div}\begin{bmatrix}-\partial_{xyy}uv\\ \partial_{xy}u\partial_x v\end{bmatrix} + \partial_{xxyy}uv \end{aligned} \tag{A.4}$$

Now adding together equations $(A.1)-(A.4)$ and integrating over Ω we get the following,

$$\begin{aligned} A_\Omega(u,v) &= \int_\Omega \{(\nabla^4 u)\,v + \operatorname{div}\begin{bmatrix}f_1\\ f_2\end{bmatrix}\} \\ \text{where } f_1 &= \nabla(\partial_x u)\cdot\nabla v - (\partial_x\nabla^2 u)\,v \\ \text{and } f_2 &= \nabla(\partial_y u)\cdot\nabla v - (\partial_y\nabla^2 u)\,v \end{aligned}$$

Following this with an application of the divergence theorem to the second term in the integral we get, for sufficiently nice Ω, with some rearrangement of terms, the Green's formula,

$$A_\Omega(u,v) = \int_\Omega (\nabla^4 u)\, v + \oint_{\partial\Omega} \{\nabla(\partial_n u)\cdot\nabla v - (\partial_n \nabla^2 u)\, v\}$$

where ∂_n denotes generalised differentiation along the outward normal.

References.

Ahlberg, J.H., Nilson, E.N. and Walsh, J.L. (1967) *The Theory of Splines and Their Applications*, Academic Press.

Aubin, J.P. (1972) *Approximation of Elliptic Boundary Value Problems*, Wiley.

Axelsson, O. (1976) Solution of Linear Systems of Equations: Iterative Methods. In *Sparse Matrix Techniques*, (ed. Barker, V.A.) Springer-Verlag.

de Boor, C. (1978) *A Practical Guide to Splines*, Springer-Verlag.

Craven, P. and Wahba, G. (1979) Smoothing Noisy Data with Spline Functions: Estimating the Correct Degree of Smoothing by the Method of Generalized Cross-Validation. *Numerische Mathematik*, 31, 377-403.

Dahmen, W. (1980) On Multivariate B-Splines. *SIAM Journal on Numerical Analysis*, 17(2) 179-191.

Dahmen, W. (1981) Approximation by Linear Combinations of Multivariate B-Splines. *Journal of Approximation Theory*, 31(3) 299-324.

Deny, J. and Lions, J.L. (1954) Les Espaces du type Beppo Levi. *Annales Institut Fourier*, 5, 305-370.

Duchon, J. (1976) Interpolation Des Fonctions De Deux Variables Suivant Le Principe De La Flexion Des Plaques Minces. *RAIRO Analyse Numerique*, 10(12) 5-12.

Duff, I.S. (1982) Research Directions in Sparse Matrix Computations. *UKAEA Harwell Research Report No. AERE-R 10547.*

Dyn, N. and Levin, D. (1982) Construction of Surface Spline Interpolants of Scattered Data over Finite Domains. *RAIRO Numerical Analysis*, 16(3) 201-209.

Dyn, N. and Levin, D. (1983) Iterative solution of systems originating from integral equations and surface interpolation. *SIAM Journal on Numerical Analysis*, 20(2) 377-390.

Eubank R.L. (1984) The Hat Matrix for Smoothing Splines. *Statistics and Probability Letters*, 2, 9-14.

Eubank R.L. (1988) *Spline Smoothing and Nonparametric Regression*, Marcel Dekker.

Hestenes, M.R. and Stiefel, E. (1952) Methods of Conjugate Gradients for Solving Linear Systems. *National Bureau of Standards Journal of Research*, 49, 409-436.

Hu C.L. and Schumaker L.L. (1986) Complete Spline Smoothing. *Numerische Mathematik*, 49, 1-10.

Hutchinson M.F. and de Hoog, F.R. (1985) Smoothing Noisy Data with Spline Functions. *Numerische Mathematik*, 47, 99-106.

Kershaw, D.S. (1978) The Incomplete Cholesky-Conjugate Gradient Method for the Iterative Solution of Systems of Linear Equations. *Journal of Computational Physics*, 26, 43-65.

Kimeldorf, G.S. and Wahba, G. (1971) Some Results on Tchebycheffian Spline Functions. *Journal of Mathematical Analysis and Application*, 33, 82-94.

Maz'ja, V.G. (1980) *Sobolev Spaces*, Springer-Verlag.

Meinguet, J. (1979a) An Intrinsic Approach To Multivariate Spline Interpolation At Arbitrary Points. In *Polynomial and Spline Approximation*, (ed. Sahney, B.N.) D.Reidel Publishing Company.

Meinguet, J. (1979b) Multivariate Interpolation At Arbitrary Points Made Simple. *ZAMP*, 30, 292-304.

Munksgaard, N. (1980) Solving Sparse Symmetric Sets of Linear Equations by Preconditioned Conjugate Gradients. *ACM Transactions on Mathematical Software*, 6(2) 206-219.

O'Connor, D.P.H. and Leach, B.G. (1979) Geostatistical Analysis of 18CC Stope Block, CSA Mine, Cobar, NSW. In *Estimation and Statement of Mineral Reserves*. Australian IMM, Melbourne, Australia, pp. 145-153.

Reid, J.K. (1971) On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations. In *Large Sets of Linear Equations* (ed. Reid, J.K.). Academic Press. pp.231-254.

Sibson, R. (1980) A Vector Identity for the Dirichlet Tessellation. *Mathematical Proceedings of the Cambridge Philosophical Society*, 87, 151-155.

Sibson, R. (1985) Nonparametric Spatial Regression. *Draft Report.* University of Bath.

Silverman B.W. (1985) Some Aspects of the Spline Smoothing Approach to Non-parametric Regression Curve Fitting. *Journal of the Royal Statistical Society, Series B*, 47(1), 1-52.

Temple, G. (1955) The theory of generalised functions. *Proceedings of the Royal Society A*, 228, 175-190.

Wahba, G. (1979) How to smooth curves and surfaces with splines and cross-validation. *Technical report no. 555*, Department of Statistics, University of Wisconsin.

Wegman E.J. and Wright I.W. (1983) Splines in Statistics. *Journal of the American Statistical Association*, 78, 351-365.

Young, D.M. (1971) *Iterative Solution of Large Linear Systems.* Academic Press.