INTERACTIVE LEAST SQUARES SURFACE FITTING

by

ANTHONY HARM SAMSOM

B.Sc. University of British Columbia, 1978

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard.

THE UNIVERSITY OF BRITISH COLUMBIA

May, 1980

(c) Anthony Harm Samsom, 1980
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of **Computer Science**

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date **May 2, 1980**
ABSTRACT

This thesis is concerned with the design and implementation of a surface fitting package in an interactive graphics environment. Surface fitting techniques are used to generate a smooth looking, easy to evaluate, bivariate function given a set of data points on some domain in the plane, and are thus useful for a variety of applications. We consider the implementation of a surface fitting technique using weighted least squares with tensor products of B-splines on regular data grids (i.e. the position of the data points can be represented as the cross product of two vectors). While somewhat more restrictive than other surface fitting methods, this technique, when applicable, is extremely efficient.

Knot placement and weight placement are discussed as methods of adapting the spline surface to rapidly varying regions on the domain. A disadvantage of the original method used to solve for the coefficients of the spline surface is that the domain of the function to be approximated must be rectangular. An algorithm to extend the surface fitting method to non-rectangular domains, thus removing this restriction, is presented. An interactive surface fitting package is provided, which allows a user to fit a spline surface to a set of data points on a regular grid. This provides a powerful tool which may be used to effectively modify the spline surface and indicate the accuracy of the approximation.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Survey of methods for Surface approximation</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Methods based upon points</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Methods based upon triangles</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Methods based upon squares</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>B-Spline Surface Approximation</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>One dimensional splines</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Generalization to two dimensions</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>Adapting the spline surface to quickly varying data and Non-Rectangular Domains</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Knot Placement</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Weight Placement</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>Relaxing Domain Restrictions</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>Results and Evaluation</td>
<td>35</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>51</td>
</tr>
<tr>
<td>A</td>
<td>A Surface Fitting Package</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>The Command Interpreter</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>The Support Routines</td>
<td>61</td>
</tr>
<tr>
<td>B</td>
<td>Generation and Evaluation</td>
<td>62</td>
</tr>
</tbody>
</table>
TABLES

Table I: Errors on function $f_1$ using evenly spaced knots 39
Table II: Errors on function $f_2$ using evenly spaced knots 40
Table III: Errors on function $f_2$ using a good knot placement 41
Table IV: Errors on function $f_3$ 41
Table V: Errors on function $f_2$ by methods from (Franke 79) 42
Table VI: Timings from generation and evaluation of spline 43
Table VII: Timings on selected methods from (Franke 79) 43
Table VIII: Errors from function $f_4$ over an even mesh 45
Table IX: Spline approx. of function $f_4$ over an optimized mesh 46
Table X: Spline approx. of function $f_4$ over an L shaped domain 47
FIGURES

Figure 1: Max-min triangle optimization

Figure 2: Smooth spline

Figure 3: Banded matrix

Figure 4: Local trouble area

Figure 5: Global trouble area

Figure 6: Non-rectangular domain

Figure 7: Function f1 (Saddle function)

Figure 8: Function f1 rotated 90 degrees

Figure 9: Function f2 (Exponential test function)

Figure 10: Function f2 (Rotated 90 degrees)

Figure 11: Function f3 (Cliff test function)

Figure 12: Plot of function f4

Figure 13: Function f4 rotated 90 degrees
ACKNOWLEDGEMENT

I would like to sincerely thank Dr. Uri Ascher for his advice and financial support in the development and writing of this thesis.
Chapter 1: Introduction

Many problems arise where the value of a bivariate function is known only at some discrete points on its domain. Surface fitting techniques use this data to generate an analytic function that approximates the unknown function. Alternatively, if the analytic representation of the original function is known but expensive to evaluate, surface fitting techniques may be used to generate a function that accurately approximates it but is much cheaper to evaluate. Surface fitting techniques are used in a variety of fields; for instance, in Graphics these techniques can be used to generate compact, accurate and smooth representations of the surfaces to be displayed. The techniques can be similarly applied in such diverse areas as Geography, Physics, Computer Aided Design and industrial milling applications (Spath 74). In this thesis we consider the implementation of one such technique using least squares fits with tensor products of B-splines on regular data grids.

In general, there are three basic building blocks used for generating surfaces: Points, Triangles and Squares. Methods using points to build surfaces are usually based on some variant of Shepard's formula (Barnhill 77). The idea behind these methods is to generate the value of an arbitrary point on the domain as a weighted average of the data points around it. Although Shepard's formula usually generates a poor fit, several variants (Barnhill 77) which improve on the approximant's continuity etc. generally do quite well. Methods based upon points are usually simple to code, have minimal preprocessing requirements and need relatively little space to represent the approximant; however, point evaluation is generally
quite expensive. Methods based upon triangles usually can provide
good surface approximations. They have relatively cheap point
evaluation cost and reasonable preprocessing time. However they
require a considerable amount of space to store the triangles and
basis functions and are usually difficult to code. Methods based upon
squares are the least flexible of the three. They adapt poorly to
non-rectangular domains and may also have trouble adapting to
irregularly spaced data. Although lookup may be faster on a
rectangular grid than on a triangular one, the basis functions over
rectangles are more complex than the corresponding ones over triangles
so point evaluation may be more expensive. However, such methods do
have distinct advantages in more restricted applications.

If the domain is rectangular in shape and the data points are
given on grid intersection points, it is possible to build a tensor
product piecewise polynomial surface over the domain. Using
B-splines, the spline surface can be constructed efficiently (de-Boor
79). The B-spline representation of the surface is compact and the
point evaluation cost is relatively inexpensive. This method of
surface fitting can be considered as the generalization of one
dimensional spline approximation to two dimensions using tensor
products.

A one dimensional polynomial spline approximation of order k
(degree < k) may be represented as a linear combination of linearly
independent basis functions where each basis function is a piecewise
polynomial of degree < k on the domain of the function. B-splines
form such a basis with each basis function having a local support. If
the number of basis functions is the same as the number of data points
(i.e. interpolation) a linear system of equations can be solved to obtain the coefficients of the spline approximant (a detailed explanation is given in Chapter 3). Using the B-spline basis results in a banded matrix which can be solved quickly. If there are more data points than basis functions a least squares fit can be generated by solving the corresponding overdetermined system of equations. It is advantageous in this case to form the normal equations in spite of conditioning problems. The matrix $A$ has a zero structure which depends upon the placement of the data relative to the knots specifying the B-splines and the order of the spline. Thus the normal equation matrix $A^TA$ has a single band whose width is dependent only on the order of the spline.

One dimensional interpolation and approximation can be extended to two dimensions using tensor products.

\[(1.2) \quad s(x,y) = \sum g_{ij} B_i(x) B_j(y)\]

where $s(x_i,y_j) = ((f(x_i,y_j), i = 1, \ldots, M) j = 1, \ldots, N)$

This construction forces the data to lie on a regular grid over a rectangular domain. Although the coefficients can be solved directly as in the one dimensional case, it is possible to solve for the coefficients by solving a series of one dimensional problems. This reduces the computational cost considerably (from $O((MN)^3)$ to $O(M^3 + MN^2 + NM^2 + N^3)$ if the matrices are full). Both systems are banded if the B-spline basis is used but the direct approach results in a large matrix with $(2(k - 1) + 1)$ bands while the one dimensional approach produces 2 small matrices with one band each. Therefore, as discussed in Chapter 3, the direct approach produces a matrix that is harder to solve efficiently.
Again a least squares fit can be generated if there are more data points than basis functions. The coefficients can be found by solving the system directly or (as in the interpolation case) solving a series of one dimensional problems. In both cases it is advantageous to form the normal equations as $A^TA$ has a much nicer zero structure than the matrix $A$. If the system is solved directly the data may be somewhat scattered (there must be at least one data point in the support of each basis function) and weights may be applied to each data point. However, if the system is to be solved as a series of one dimensional problems, the data must be on a regular grid and the weights may only be applied to these grid lines instead of the individual data points.

The one dimensional approach imposes some restrictions on the adaptability of the surface to trouble areas (rapidly changing data in some region) and its applicability to non-rectangular domains. Knot placement is restricted to changing knot lines (because of the way the basis functions are constructed). As a result the adaptability of the spline to trouble spots on the domain by changing the knots is also limited. There are some reasonable knot placement algorithms for one dimensional problems. An algorithm proposed by Dodson (Dodson 72) is discussed and an extension to the two dimensional problem is proposed. The method is applied to two dimensions by having the user interactively choose a "scan line" on the domain which the one dimensional knot selection technique is applied to. The way that weights can be applied to the domain implies similar problems in using weights to force the spline surface to adapt to trouble areas. No method is proposed for the automatic application of weights. Weights are probably most effective when applied carefully by the user in an
interactive environment where the results can be carefully monitored. In both knot placement and the application of weights the use of a graphics display in an interactive environment greatly improves the effectiveness of the method.

This approach to surface fitting is restricted to rectangular domains. Since many problems are posed over non-rectangular domains, it would be nice to extend the method to include them. The approach suggested is to fit a spline surface over a rectangular domain which encloses the domain of the function to be approximated. This leaves holes in the domain which should be "filled" with data that has a minimal effect on the fit of the spline. The method proposed to generate this data consists of two stages:

a) Filling the holes using a Coons' patch defined on a rectangle around each hole in the domain.

b) Generating a (possibly weighted) surface over the domain and using the values of the surface over the holes to update the values generated by the Coons' patch.

A surface fitting package that allows the user to interactively fit a polynomial spline surface to a set of regularly spaced data points is presented. The graphics capabilities provide a powerful tool a user can use to adapt the spline fit and evaluate the results. The package is divided into two sections: the command interpreter and the support routines. The command interpreter is written in Pascal and the support routines are in Fortran. This guarantees portability restrictions. However, as the package is highly dependent upon UBC graphics routines, it is unreasonable to expect portability anyway.

The thesis is divided into five chapters and three appendices,
Chapter 1 being the introduction. Chapter 2 contains a survey of some of the popular surface fitting techniques. A development of the generalization of one dimensional splines to two dimensions is presented in Chapter 3. Chapter 4 contains two short sections: Adapting the spline by moving the knots and/or changing the weights and adapting the spline to non-rectangular domains. Chapter 5 contains some results from the surface fitting package and some limited comparisons with methods presented in (Franke 79). The conclusions are stated in Chapter 6. Appendix A contains a descriptions of the surface fitting package and Appendix B contains the source code for the two routines that generate and evaluate the spline surfaces.
Chapter 2: Survey of methods for surface approximation

The problem of surface approximation arises frequently in practice. Given a set of data points \( F_i = f(x_i, y_i) \), \( i = 1 \ldots N \) in some domain \( D \), one may wish to construct a smooth-looking surface \( s(x, y) \) which approximates the function \( f \) on \( D \). The approximation \( s(x, y) \) should be easy to store, evaluate, differentiate, integrate etc. Even if the function \( f \) is known everywhere in \( D \), if \( f \) is complicated one may want to replace it by the nicer function \( s \). There are three basic building blocks for generating surfaces: POINTS, TRIANGLES and SQUARES. This chapter contains a brief survey of some popular methods of generating surfaces using these building blocks with some discussion of the properties, advantages and disadvantages of each method.

a) Methods based upon points.

The basic idea behind Shepard's formula and its generalizations (Barnhill 77) is to let the value of the approximating function at an arbitrary point \( (x, y) \) be defined by some weighted average of the data points in \( D \). The data points close to \( (x, y) \) influence \( s(x, y) \) more than points farther away. Shepard's formula can be written as:

\[
SF = \begin{cases} 
\sum_{i=1}^{N} \frac{F_i}{d_i^2} & (x, y) \neq (x_i, y_i) \\
\sum_{i=1}^{N} \frac{1}{d_i^2} & (x, y) = (x_i, y_i) \\
F_i & (x, y) = (x_i, y_i) 
\end{cases}
\]

\[ (2.1) \]
Where \( d_i \) = the distance from the i'th data point to \((x, y)\).

\[
(2.2) \quad d_i = \sqrt{((x-x_i)^2 + (y-y_i)^2)}
\]

\( \mu = \text{constant} > 0 \)

This can be rewritten as

\[
(2.3) \quad SF = \frac{\sum_{i=1}^{N} W_i F_i}{\sum_{i=1}^{N} W_i}
\]

where

\[
W_i = \prod_{j=1}^{N} \frac{1}{d_j^u}
\]

The slope, smoothness and continuity of the surface depends upon \( u \). If 0 < \( u \) < 1 then there are cusps around the data points. If \( u = 1 \) there are breaks (jump discontinuities in \( f'(x,y) \) at \((x_i,y_i)\)) and for \( u > 1 \) there are flat spots where

\[
(2.4) \quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = 0
\]

Barnhill in (Barnhill 77) suggests \( u = 2 \) as a good compromise (for an aesthetically pleasing surface).

One possible generalization of Shepard's formula is to force the surface to interpolate the first partial derivatives at each data point as well as the value at the data points:
This function provides a $C^1$ continuous fit to the data.

Since every data point contributes to the value of $s(x,y)$ the cost of one point evaluation of $s(x,y)$ is quite high relative to other methods for surface approximation (for reasonable $N$). Franke and Little (Barnhill 77) have proposed a simple scheme to localize support. The scheme consists of replacing $1/d_i$ in (2.1) or (2.5) by

$$
(2.6) \quad \left[ \frac{(R_i - d_i)_+}{R_i d_i} \right]^2
$$

where $R_i$ = radius of support around $(x_i,y_i)$

$$(R_i - d_i)_+ = \begin{cases} 
R_i - d_i & (R_i - d_i) \geq 0 \\
0 & \text{otherwise}
\end{cases}$$

The cost per point evaluation is still high (for a reasonable fit) but if the number of data points in the domain is large, this considerably reduces the number of operations per evaluation.

$SF$ and $S^1F$ are rational functions. They do not however fit polynomial functions very well: since

$$
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = 0
$$

for $SF$, it cannot model a linear function exactly. Using a similar argument it can be shown that $S^1F$ cannot model a quadratic polynomial precisely. Various researchers have modified Shepard's formula to
increase function precision relative to polynomials. (Maclean 74) (Barnhill 77).

In (Barnhill 77) Barnhill shows how to use the following theorem to increase function precision.

The boolean sum \( P \oplus Q = P + Q - PQ \) has at least the interpolatory properties of \( P \) and the function precision of \( Q \).

Barnhill uses the theorem to construct two hybrid surface interpolants in his paper (Barnhill 77). \( P \) is replaced by Shepard's formula and \( Q \) is replaced by either a least squares fit or a 9 degree triangular interpolant. In both cases, the form of \( Q \) did not place restrictions on the domain. The cost of increasing the function precision this way is high, however, and the choice of \( Q \) may place restrictions on the shape of the domain or on the position of the data points or both.

Shepard's formula and most of its variants do not have any restrictions on the shape of the domain or placement of the data points and the methods are generally easy to implement. However, approximant evaluation is expensive as is increasing its precision (higher derivatives etc.). Information about the shape of the domain and its boundaries is not implicit to the method and must be kept separately.

b) Methods based upon triangles.

Using methods based on triangular elements to build a surface approximation is probably more complex to implement than other methods. Therefore, much more programming effort is required to build
a system using this approach. Methods based upon triangles are usually divided into two stages:

1) Preprocessing where triangulation of the domain optimization of the triangles and generation of the function defined over each triangle are handled.

2) Function evaluation where for an arbitrary point \((x,y)\) the triangle which contains the point is found and the function defined over that triangle is evaluated to obtain \(s(x,y)\).

1) Preprocessing

There are several algorithms available to construct the triangular grid (Lawson 77) (Fowler and Little 79). Lawson suggests the following algorithm (for \(D\) convex).

a) Choose a point \(p^*\) on the boundary of \(D\) having the smallest \(X\) coordinate.

b) Sort the remaining points in \(D\) in increasing Euclidian distance from \(p^*\). Let \(p^* = p_1\) and denote the remaining points as \(p_2, p_3, \ldots, p_n\).

c) The first edge is drawn connecting \(p_1\) and \(p_2\). The next point in the sequence not collinear with \(p_1\) and \(p_2\) is connected with \(p_1\) and \(p_2\) to form the first triangle. If the third point of this triangle is not \(p_3\) then relabel it as \(p_3\) and shift all others to \(p_k\) up by 1.

d) Build an initial boundary list consisting of \(p_1, p_2, p_3\), and a second occurrence of \(p_1\) along with their angular coordinates. Let \(C\) be the centroid of triangle \((p_1, p_2, p_3)\); then the angular coordinates of a point \(p_k\) is measured as the angle between the half ray starting at \(p_1\) through \(C\) and the half ray from \(C\) to \(p_k\). The
angle is measured counter clockwise around C and is always between 0 and 2\pi. The second occurrence of p_1 in the boundary list is assigned the angular coordinate 2\pi.
e) For each point p_k, k = 4, ..., N
i) Determine the angular coordinate of p_k.
ii) Find a pair of boundary points whose angular coordinates bracket this angle.
iii) Attach p_k to these points and place edge opposite p_k of new triangle on a stack.
iv) If the stack is not empty unstack one edge and apply the local optimization procedure to it. If the edge is to be swapped, do so and place the two edges opposite p_k on the stack. Iterate until the stack is empty.
v) try to connect p_k to another neighboring boundary point. If possible goto iii otherwise k := k+1 and goto i.

The processing cost is estimated by Lawson to be O(n^{4/3}) + O(n \log n).

There are several local optimization techniques available. The purpose of these is to prevent any angles in the triangular grid becoming small. If some angles are allowed to become small this causes accuracy problems in the derivatives: See (Strang and Fix) or (Mitchell and Waite). A comparison of several local optimization methods is given in (Lawson 77). The max-min angle criterion is discussed here.

The basic idea in the max-min angle criterion is to increase the size of the smallest angle in the triangular grid. This method specifically attempts to maximize this minimum angle locally.
Consider the following figure:

![Max-min triangle optimization](image)

If the smallest angle is increased by replacing the edge between $T_1$ and $T_2$, the swap is made creating two new triangles as shown above.

Once the triangular grid has been built on the data points, generation of the functions defined over the triangle can proceed. Since, for interpolation, the basis functions are usually chosen so that they are independent over each triangle (i.e. the basis is local) no system of equations need be solved to obtain the surface. As a result, if the functions are simple (linear etc.) the generation of the surface may be postponed until the value of $s(x,y)$ is asked for in that particular triangle. The following is an example of the construction of a piecewise linear surface over the triangular grid.

The surface can be considered to be constructed of $N$ piecewise linear basis functions

$$s(x,y) = \sum_{j=1}^{N} F_j B_j(x,y)$$

where

$$B_j(x_i,y_i) = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases}$$

Any triangle in the grid will have three non-zero basis functions defined on it. These basis functions are defined on the canonical triangle as follows:
So the value of \( s(x,y) \) on this triangle is given by

\[
(2.9) \quad s(x,y) = F_1 (1 - u - v) + F_2 u + F_3 v
\]

((x,y) is in \( T \) with values \((F_1, - , F_2, - , F_3, - )\) at the vertices)

Since any triangle in the domain can be mapped onto the canonical triangle using the following transformation:

\[
(2.10) \quad u = \frac{(x - x_1)(y_3 - y_1) - (y - y_1)(x_3 - x_1)}{\text{det}(J)}
\]

\[
v = \frac{(y - y_1)(x_2 - x_1) - (x - x_1)(y_2 - y_1)}{\text{det}(J)}
\]

where

\[
J = \begin{pmatrix}
(x_2 - x_1) & (x_3 - x_1) \\
(y_2 - y_1) & (y_3 - y_1)
\end{pmatrix}
\]

The value of any point in the domain can be calculated by:

a) Finding the triangle which contains it.

b) Mapping the point onto the canonical triangle and using (2.3), where \( F_1, F_2, F_3 \) are the values of the function at the vertices of the triangle.

A number of functions suitable for surface approximation over triangles are given in (Barnhill 77). To get \( C^1 \) continuity, first derivatives must be approximated. Since for higher degree \( C^0 \) and \( C^1 \) approximation the coefficients are expensive to generate, they are usually calculated and stored during the preprocessing stage.

The cost per function evaluation depends on two factors: how fast
the triangle which contains the point to be evaluated can be found and the complexity of the basis functions. A reasonable search time (Lawson 77) is $O(\log_2 N)^2$ for an arbitrary point in the domain. The piecewise linear surface involves around 15 multiplications per function evaluation plus search time. The cost per function evaluation is generally considerably better here than most variations of Shepard's formula (for reasonable accuracy). If not all the data points are used to generate the surface, this approach can be made to adapt to local "trouble spots" in the domain. In (Fowler and Little 79) this was done by first generating a triangular grid using only the local maxima and minima on the domain. Then if the surface over a given triangle did not meet its accuracy requirements a point was chosen interior to this triangle and the triangle was replaced by three new triangles. This process is repeated until each triangle meets its accuracy requirements. When compared to the piecewise linear fit generated by using all the data on the rectangular grid good results were obtained using this method in terms of data compaction.

If the user has a package available or can afford to build one this is the method of choice when working with large numbers of data points on arbitrary domains and/or the data points are not regularly spaced.

c) Methods based upon squares. (rectangles)

Rectangles are the least flexible of the three methods. They have domain restrictions and are not easily adaptable to irregularly spaced data. However if the domain is suitable and the
data points are regularly spaced (or can be moved to grid lines using another method), this method has definite computational advantages.

Most interpolation methods defined over rectangles can be obtained from "transfinite" Coons' patches developed by S. A. Coons in (Coons 64). The bilinearly blended Coons patch defined on the unit square is given by:

\[
(2.11) \quad s(u,v) = [-u, u] \begin{bmatrix} 1-v \\ v \end{bmatrix} + \begin{bmatrix} [F(0,0), F(0,1)] \\ F(1,1) \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix} - [-u, u] \begin{bmatrix} F(0,0) & F(0,1) \\ F(1,0) & F(1,1) \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}
\]

where \((0 < u, v < 1)\)

This patch interpolates to the data all around the unit square. If \(F(0,v)\) is replaced by the discretization

\[
(2.12) \quad F(0,v) = (1-v) F(0,0) + v F(0,1)
\]

a bilinear interpolant results. Similarly quadratic, cubic, and other interpolants can be generated using the appropriate discretization. The discretization of this patch generate only \(C^0\) interpolants however. Patches which when discretized give \(C^1\) or better surfaces exist. References may be found in (Barnhill 77).

If the domain of \(f(x,y)\) is rectangular in shape and the data points exist on grid intersection points then it is possible to build a tensor product polynomial surface. Although the construction and solution of a linear system of equations is necessary, the local support offered by an appropriate basis such as B-splines results in a system of equations which can be stored
and solved efficiently. Using this approach, obtaining $C^1$ continuity or better is cheap as is point evaluation of $s(x,y)$ and storage of the surface representation. It is also possible to efficiently generate $s(x,y)$ as a weighted least squares fit to data which is usually difficult to accomplish using other approaches to surface fitting.
Chapter 3: B-Spline surface approximation.

This approach to surface approximation can be considered as the generalization of one dimensional spline approximation to two dimensions via tensor products. One dimensional polynomial splines are quite popular as they are both computationally efficient and quite flexible for most applications. They are also well understood mathematically.

a) **One Dimensional splines:**

Let \( \Pi \) be a partition \( a = t_0 < t_1 < \ldots < t_M = b \), then a (polynomial) spline function of order \( k \) (degree \( k-1 \)) with \( N - 2 \) internal knots is a function \( s(x) \) where \( s(x) \) is a polynomial of degree \( < k \) on each interval \( ((t_i, t_{i+1}) \ i=0 , \ldots , M-1) \). \( s(x) \) is said to be a smooth spline if \( s(x) \) is in \( C^{k-2}[a,b] \). Let \( P_k, \pi \) denote the finite dimensional linear space of piecewise polynomials defined on \( [a,b] \) having breakpoints \( \Pi \) and having degree \( < k \).

Using the definition of a smooth spline, it is fairly easy to construct a spline interpolant to a given set of data.

Example: Smooth cubic spline interpolation.

![Smooth spline](image)

\( s(x) \) is a piecewise cubic polynomial which means:

\[
(3.1) \quad s(x) = s_i(x) = a_i(x-t_i)^3 + b_i(x-t_i)^2 + c_i(x-t_i) + d_i
\]
since \( s(x) \) is in \( C^2[a,b] \):

\[(3.2)\]

1) \( s_i(t_{i+1}) = s_{i+1}(t_{i+1}) = f(t_{i+1}) \)

\[= a_i h_i^2 + b_i h_i^3 + c_i h_i + d_i = d_{i+1} \]

2) \( s'_i(t_{i+1}) = s'_{i+1}(t_{i+1}) \)

\[= 3a_i h_i^2 + 2b_i h_i + c_i = c_i \]

3) \( s''_i(t_{i+1}) = s''_{i+1}(t_{i+1}) \)

\[= 6a_i h_i + 2b_i = 2b_i \]

where \( h_i = (t_{i+1} - t_i) \)

This implies that we need to solve a system of equations with \( N+2 \)
unknowns in \( N \) equations to obtain all the coefficients. In practice,
the two additional unknowns are usually provided by either setting
\( s''(a) = s''(b) = 0 \) (natural spline) or setting \( s'(a) \) and \( s'(b) \) to some
approximation of the slope at the end points of the interval. Note
that the above equations form a tridiagonal system which is to be
solved for the coefficients \( c_i \), \( i=1,M \).

This approach, although simple for this example is not easily
applied to least squares problems or generalized to two dimensions.
Alternatively the problem can be re-stated as

\[(3.3)\]

\[ s(x_i) = \sum_{j=1}^{M} a_i B_i(x_j) \quad i = 1, \ldots, M \]

where each \( B_i \) is a piecewise polynomial in \( P_4, \pi \) and \( B_i \) is in \( C^2[a,b] \).
Solving the resulting system of equations for \( a_i \) would generate \( s(x) \).
Clearly any linearly independent polynomial basis in \( P_4, \pi \) would
suffice, but if the basis functions are chosen carefully the system
could become as computationally attractive as the previous approach.
A poorly chosen basis has, along with the added computation involved
in solving a full matrix, conditioning problems as well. One basis
with the desired local support is the B-spline basis.

\[(3.4) \quad B_i(x) = [t_i, \ldots, t_{i+k}](t-x)^{k-i} \]

where \((t-x)_+^{k-i} = \max (t-x,0)^{k-i}\)

and \([t_i,\ldots, t_{i+k}]f\) denotes the \(k\)-th divided difference of function \(f\) defined recursively here as

\[(3.5) \quad [t_i, t_{i+1}]f = \frac{f(t_i) - f(t_{i+1})}{t_i - t_{i+1}} \]

\([t_i,\ldots, t_{i+k}]f = [t_i,\ldots, t_{i+k-1}]f - [t_{i+1},\ldots, t_{i+k}]f \]

\[\frac{t_i - t_{i+k}}{t_i - t_{i+k}} \]

It is easy to show that \(B_i\) has local support.

if \(x > t_{i+k}\) then \((t-x)_+ = 0 \Rightarrow B_i(x) = 0\)

if \(x < t_i\) then \((t-x)^{k-1}\) is a polynomial of degree \(k-1\) and since

\([t_i,\ldots, t_{i+k}]f = f^k(\zeta)\) where \(\zeta\) is some point on the line

\(B_i(x) = 0\).

Using Leibniz' formula which states that if \(f(x) = g(x)h(x)\) for all \(x\) then

\[(3.6) \quad [t_i,\ldots, t_{i+k}]fg = \sum_{r=i}^{i+k} [t_i,\ldots, t_r]f \cdot [t_r,\ldots, t_{i+k}]g \]

we can define the recurrence relation

\[(3.7) \quad [t_i,\ldots, t_{i+k}]f = \frac{x - t_i}{t_{i+k} - t_i} [t_i,\ldots, t_{i+k}]f \]

\[+ \frac{t_{i+k} - x}{t_{i+k} - t_i} [t_i,\ldots, t_{i+k}]g \]

which can be used to efficiently generate B-splines when needed for arbitrary \(k\) (de-Boor 79).

B-splines over an even mesh have some very nice properties which lead to a well conditioned system of equations to solve (de-Boor 79).
For \( k=4 \) the corresponding B-spline basis, \( B_4(x) \), on an even mesh is given by

\[
S((x - x_i)/h) \quad (h \text{ is the distance between mesh points}).
\]

\[
S(x) = \begin{cases} 
0 & x \leq -2 \\
\frac{(2 - x)^3}{24} - \frac{(1 - x)^3}{6} - \frac{x^3}{4} + \frac{(1 + x)^3}{6} & -2 \leq x \leq -1 \\
\frac{(2 - x)^3}{24} - \frac{(1 - x)^3}{6} - \frac{x^3}{4} & -1 \leq x \leq 0 \\
\frac{(2 - x)^3}{24} - \frac{(1 - x)^3}{6} & 0 \leq x \leq 1 \\
\frac{(2 - x)^3}{24} & 1 \leq x \leq 2 \\
0 & x \geq 2
\end{cases}
\]

(3.8) \quad S(x) = \begin{cases} 
0 & x \leq -2 \\
\frac{(2 - x)^3}{24} - \frac{(1 - x)^3}{6} - \frac{x^3}{4} + \frac{(1 + x)^3}{6} & -2 \leq x \leq -1 \\
\frac{(2 - x)^3}{24} - \frac{(1 - x)^3}{6} - \frac{x^3}{4} & -1 \leq x \leq 0 \\
\frac{(2 - x)^3}{24} - \frac{(1 - x)^3}{6} & 0 \leq x \leq 1 \\
\frac{(2 - x)^3}{24} & 1 \leq x \leq 2 \\
0 & x \geq 2
\end{cases}

For an uneven mesh and a given \( k \) (3.7) can be expanded to form the basis functions explicitly. However each basis function cannot be generated by scaling a "canonical basis function" as is done here for an even mesh. For an arbitrary \( k \), equation (3.7) is used implicitly to generate the basis functions.

There are some problems defining the B-splines at the end points of the interval in question using the current partition. For the remainder of the thesis a partition on \([a,b]\) will be defined as

\[
t_1 \leq \ldots \leq t_k = a \leq t_{k+1} \leq \ldots \leq t_M = b \leq \ldots \leq t_{N+k}
\]

Note that no more than \( k \) knots may coalesce or the system becomes rank deficient. If two knots coalesce at \( x_j \) the spline approximation loses another degree of smoothness at that knot, if \( k \) knots coalesce at \( x_j \) there is a jump discontinuity at that point. The knot sequence above applied to \( N \) data points specifies a \( N \times N \) banded system of equations to be solved. Solving the system
for the coefficients (from 3.4) provides the interpolating spline \( s(x) \).

This approach can be modified quite readily so that \( s(x) \) approximates the data instead of interpolating it. Frequently in applications there are more data points than B-splines (an overdetermined system of equations).

let \( I_M \) = number of data points

and let \( I_M > M \)

then for the least squares approximation, the system

\[
(3.11) \quad (A^T A)c = A^T y \text{ where } c = [a_1, a_2, \ldots, a_M]^T
\]

define the normal equations which can be solved quite efficiently using a direct method for solving banded symmetric positive definite systems of equations. Since from (3.6) \( B_i \) has support over \( k \) intervals:

\[
(3.12) \quad B_i B_j \neq 0 \text{ iff } |i - j| < k
\]

the system is banded (for \( k = 4 \Rightarrow \text{bandwidth} = 5 \)) and diagonally dominant (for an even mesh).

Forming the normal equations in this case, although providing a nice matrix, could cause some conditioning problems for uneven meshes as \( K(A^T A) = K(A)^2 \) (where \( K(B) \) denotes the condition number of a matrix \( B \)) and \( K(A) \) can get quite large if an abruptly varying mesh is used. A knot sequence that results in a high condition number can be easily specified. If the knot sequence is spaced so that some B-splines have many data points in their support and some have very few (and these
barely within the support of that B-spline), then there will be large
and very small diagonal elements in $A^T A$. However, the zero structure
of the matrix $A$ depends on the placement of the data and knots, so
solving and storing it as efficiently as $(A^T A)$ is difficult. This
(for most problems) makes the solution using the normal equations
quite attractive. For B-splines the practical experience indicates
that an uneven mesh will not cause accuracy problems. It appears that
$K(A)$ is a poor indication of what is happening for this system.

b) Generalization to two dimensions

Generalization to two dimensions using tensor products is the
next step in the process. A two dimensional basis function is
constructed quite easily from B-splines using tensor products:

$$ (3.13) \quad \psi_{ij}(x,y) = B_i(x) \, B_j(y) $$

Splines of different orders may be used in the $x$ and $y$ direction and
we denote the orders of the spline in the $x$ and $y$ directions as $k_x$ and
$k_y$ respectively. A spline surface approximating the data could be
characterized using this basis as:

$$ (3.14) \quad s(x,y) = \sum_{i,j=1}^{MxN} g_{ij} B_i(x) B_j(y) $$

Note that when interpolation is used, this construction implies that
the data lies on a "wire frame" grid over a rectangular domain.

Obtaining the coefficients for $s(x,y)$ using interpolation is
quite straightforward. There are $MN$ equations in $MN$ unknowns so all
that has to be done is to solve a system of equations $MN \times MN$ in size.
Because of the local support the B-spline basis offers, the matrix is
again banded:
$k_x$ and $k_y$ are assumed to be the same ($=k$) to simplify the description but because there is more than one band, (the number of bands depend on $k_x$ and $k_y$, and the distance between the bands depends on $M$ and $N$), it is harder to store and solve the system efficiently than in the one dimensional case.

It is possible, however, to solve this system of equations by solving a series of corresponding one dimensional problems in each direction. Let $G_{M \times N}$ be the matrix of coefficients ($G_{ij} = g_{ij}$) and let $L$ be the matrix containing the values of the function to be interpolated ($L_{ij} = f(x_i,y_j)$). The matrices $A$ and $B$ are the matrices corresponding to the B-splines on each axis (i.e. $A_{ij} = B_i(x_j)$ and $B_{ij} = B_i(y_j)$). From (3.14)

\begin{equation}
(3.15) \quad s(x_k,y_l) = \sum_{i=1}^{M} \sum_{j=1}^{N} g_{ij} B_i(x_k) B_j(y_l) = L_{kl}
\end{equation}

\begin{align*}
&= \sum_{i=1}^{N} \left( \sum_{j=1}^{M} g_{ij} B_i(x_k) \right) B_j(y_l) \\
&= \sum_{j=1}^{N} (A(k) \cdot G^{(1)}) B_j(y_l)
\end{align*}

\begin{equation}
(3.16) \quad s(x_k,y_l) = (A \cdot G \cdot B^T)_{kl}
\end{equation}

Solving $A \cdot G \cdot B^T = L$ for $G$ is done in two steps:
(3.17)

1) \( A X(j) = L(j) \quad j = 1, \ldots, N \Rightarrow X = G B^T \Rightarrow X^T = B G^T \)

2) \( B G^T(j) = X^T(j) \quad j = 1, \ldots, M \)

where \( X(j) \) is the \( j \)'th column of \( X \) etc.

Since \( A \) and \( B \) are banded (1 band) matrices of size \( M \times M \) and \( N \times N \), the storage and computation requirements are reduced considerably over directly solving the system. De-Boor in (de-Boor 79) estimates the complexity of the direct approach as \( O((MN)^3) \) and the separated system as \( O(M^3 + MN^2 + NM^2 + N^3) \) for a general basis (full matrices) but the difference (for small \( k_x \) and \( k_y \)) is even larger using a local basis, as the second system can be solved and stored more efficiently.

We consider again the problem where there are more data points than approximating functions. If the direct approach is used, an \( MN \times MN \) system of equations again has to be solved to obtain the coefficients when the normal equations are used. We no longer have the strict constraints on the placement of the data, only the constraint that there must be a data point in the support of each basis function (where the function is non-zero). It is again advantageous to form the normal equations, \( A^T A x = A^T b \), in spite of possible conditioning problems as the zero structure for \( A^T A \) is dependent only on \( M, N \), and the support of the basis functions which makes the matrix easier to store and manipulate. The matrix is banded with bands separated by either a distance of \( M \) or \( N \) depending on the ordering.

If the data points are on a rectangular grid, the problem can again be reduced to solving a series of one dimensional problems as in
equation (3.17).

\[ (3.18) \quad A^T A B B^T = A^T L B \]

which can be solved (as for interpolation) in 2 steps:

1) \[(A^T A) x(j) = (A^T L)(j) \quad j = 1, \ldots, L_n \]

2) \[(B^T B) G^T (j) = (B X^T)(j) \quad j = 1, \ldots, M \]

which is equivalent to (3.16) given the same data. From (3.16) \( A^T A \) and \( B^T B \) are banded matrices (1 band each) a property which can also be used to advantage in constructing \( A^T A \) efficiently. Although theoretically there could be problems because of the large condition number that this system of equations could have with an uneven mesh, no difficulties have arisen in practice.

Weights can be added to this process as in the larger system

\[ (3.19) \quad A^T W_x A B W_y B^T = A^T W_x L B W_y B^T \]

where \( W_x \) and \( W_y \) are diagonal matrices specifying the "confidence" in each grid line in \( x \) and \( y \). This is considerably different than \( A^T W x X = A^T W b \) in the full system where each weight applies to a single data point.
Chapter 4: Adapting the Spline Surface to Quickly Varying Data and Non-Rectangular Domains

Least squares spline surfaces generated using the separation method described in Chapter 3 are quite inexpensive to construct and evaluate. However, they have some limitations on their ability to adapt to "trouble" areas (where the surface to be approximated changes abruptly) and their ability to adapt to non-rectangular domains. To change the shape of the spline surface either weights may be added or the knot spacing may be changed. We discuss the knot placement first.

a) Knot Placement

Changing the knot sequence in each direction can in some cases dramatically improve the surface approximation. In other cases, however, since only grid lines can be moved, changing the knot sequence may be ineffective.

If we have a surface to be approximated which contains a single local trouble spot, knots can be added until the desired accuracy is reached in that region.

![Figure 4: Local trouble area](image)

Notice that more than just the trouble spot receives added attention.

However, if the trouble spot is not local to one area and not parallel to one of the axes, it may be impossible to adapt the mesh efficiently.
Only increasing the number of knots everywhere will improve the accuracy in this example. This time the whole surface gets a great deal more attention, including areas where it is not needed.

Despite the limitations, the approximation to many surfaces may be improved by a careful placement of knots. Since only knot "lines" can be moved, it is possible to use one dimensional knot optimization methods along these lines for certain problems. There are two basic approaches to knot placement in one dimension:

a) Optimal knot placement solved as a nonlinear optimization problem (Jupp 78). This is expensive and the results are not much better than

b) Choosing a good knot placement using an approximation of the error across the interval.

One approach to finding a good knot placement is outlined below, for more details see (Dodson 72). The distance from $f(x)$ to $s(x)$ in the $L_2$ norm is given by:

\[
\|f - s\|_p = \left( \int_a^b |f - s|^p \, dx \right)^{1/p}
\]
Dodson presents the following approximation to this distance:

\[
(4.2) \quad \| f - s \|_p \leq C \int_a^b |f^k(t)|^\sigma \, dt
\]

where \( \sigma = \left( \frac{[k + 1]}{p} \right)^{-1} \)

Equalizing the distance from \( f(x) \) to \( s(x) \) across each subinterval so that

\[
(4.3) \quad \int_{t_i}^{t_{i+1}} |f^k|^{\sigma} \, dt = \frac{1}{n-k} \int_a^b |f^k|^{\sigma} \, dt
\]

should provide a good knot placement.

Dodson suggests the following as an estimate for \( f^k(t) \) as it is unavailable for most applications:

a) Generate a spline approximation \( s(x) \) to the data with some preliminary knot placement with the desired number of knots (usually provided by the user).

b) Differentiate \( s(x) \) \( k-1 \) times to obtain the piecewise constant function \( s^{k-1}(x) \).

c) Join the midpoints of each adjacent piecewise constant in \( s^{k-1}(x) \) to form a new piecewise linear function \( f^{k-1}(x) \).

d) Differentiate \( f^{k-1}(x) \) to obtain the piecewise constant function \( f^k(x) \).

This approximation to \( f^k(x) \) is used in (4.3) to generate the desired mesh.

If the original spline fit \( s(x) \) approximates the function \( f(x) \) reasonably well then using the new knot placement obtained above will
generally improve the fit of the spline function. However if $s(x)$
does not fit the data well to begin with, the new knot sequence may
not be an improvement over the old and the error $\|f - s\|_p$ may grow.

Applying this or any other one dimensional knot selection
technique to two dimensional splines in an automatic manner is
difficult if not impossible to do effectively. If the problem has a
local trouble spot (Figure 4) then if the scan lines for the one
dimensional optimization are chosen correctly, the knots suggested by
the optimization technique (along these lines) will improve the
overall surface approximation. If there is more than one trouble spot
it is not so obvious how to apply the optimization techniques and if
the situation resembles Figure 5 then the techniques are useless.

It appears unlikely that an efficient and reliable algorithm can
be developed to choose good knot sequences for a two dimensional
surface fitting problem with spline products. It is much more
reliable and efficient to have a user controlling the position of the
knots and deciding when the fit is acceptable (or when its time to
give up). In the package described in Appendix A, a user can opt to
apply the one dimensional optimization techniques to suggest knot
sequences to be used for his problem. All that is intended is to
provide a tool to suggest knot sequences which can be altered
interactively by the user or (more likely) completely ignored. The
most powerful tool available to generate a good spline fit is the
ability to interactively alter the spline and to see the results.
b) **Weight Placement**

Weights can also be applied to the surface to alter the shape of the spline approximation. They can be used in two ways, either to force a closer fit in one area by increasing the weight in that area or to ignore the effect of some "impossible" area (sacrificing accuracy there to increase accuracy on the rest of the surface). Weights must be applied conservatively, however. Improperly applied they can cause severe conditioning problems in the system which, unlike the large condition number caused by an irregular mesh, may have a severe effect on the accuracy of the computed \( s(x, y) \).

As stated in Chapter 3, weights cannot be applied to individual data points if equation (3.19) is used to obtain the coefficients. The weights can only be applied to grid lines. This causes problems similar to those encountered with the knot placement. If the problem resembles Figure 4 it is not possible to apply weight to the trouble spot without applying it along the grid lines outside the trouble area (a nonlocal weighting results). If the problem is not local as in Figure 5, then the addition of weights will not help the fit. Providing a system that automatically generates the weights would not be very efficient. Allowing a user to interactively add, delete and change weights, however, is a very effective approach to the problem.

c) **Relaxing Domain Restrictions**

Because of the nature of our approximating function \( s(x, y) \), it has a very severe restriction on the shape of the domain. Since the method has distinct computational and/or space advantages over more
general surface fitting techniques, it would be nice to be able to efficiently adapt this method to non-rectangular domains. Unfortunately, for a general domain $D$, no such technique exists but it is still possible to extend this method to nearly rectangular domains.

For many problems, if a region does not conform to the shape of the basis functions, the general technique is to apply some transformation $T$ to the domain to map it onto a more suitable domain. The method has a major drawback for our application however, as it also moves the data points off the grid lines. The data points must therefore be resampled using a more general surface fitting method before our approach may be taken. The transformation, if complex, could also add considerable point evaluation and preprocessing cost.

If instead, a rectangle $D'$ is superimposed over $D$, it is possible to generate a surface over $D'$ providing that data is given in $D' \cap D$ and the data lies along mesh lines. The cost per point evaluation is increased however as checking whether or not a point $(x,y)$ is in $D$ must be done for each point evaluation.

![Diagram](image)

**Figure 6: Non-rectangular domain**

The problem now is how to generate data points in $R$ which will not adversely affect the spline fit on $D$. If it were possible to weigh each data point individually, then $R$ could be ignored but because of the way $s(x,y)$ is generated weights can only be applied to
mesh lines. Good data must be generated to fill R if a reasonable fit is to be expected. To facilitate a good fit the boundary between D and R should be smooth (possibly higher than \( C^0 \)). While there may be many different methods of attempting this goal we have chosen the following:

a) For each region R in D find the smallest rectangle which encloses R.
b) Use a variant of a bilinear Coons' patch over this rectangle to generate data inside R.
c) Generate a (possibly weighted) surface \( s(x,y) \) over D. Use the values from \( s(x,y) \) to replace the data values in R if the original fit is not satisfactory (otherwise quit).
d) Repeat (c) (possibly reducing the weights) until the iteration does not improve the fit or the fit is satisfactory.

If the region R is not rectangular, a weighted surface approximation to reduce the effect of generated data on D may be undesirable since it will also affect the fit along the boundary between D and R inside D.

The rectangles enclosing R and the discretized Coons' patch are obtained in the following manner.

i) First every point on \( D' \) is visited, if \((x_i,y_j)\) is not in D then its position is recorded in two weight vectors at \( WX_i \) and \( WY_j \).

ii) For each region the following is done:

a) If one side of R lies on the boundary of the rectangle, data on that side is given by the straight line interpolation between the two end points on that side in D just outside R.
b) If two adjacent sides are on the boundary of the rectangle,
the vertex (corner of D') value is first computed as the average of the two nearest corners of the rectangle around R and (a) is applied to both edges.

c) The Coons' patch is then discretized by letting the function values on the edge of the unit square be the data values on the rectangle enclosing R. The discretized Coons' patch is then evaluated at each point inside R.

Generating data this way has varied results. Since for a non-rectangular R there are real function values inside the rectangle enclosing R there may be a jump discontinuity at the boundary between R and D if the real data in the area does not agree with the Coons' patch approximation. If the surface around R is smooth the spline fit may not be adversely affected since the discontinuity should be small. However if the surface moves quickly around R then the discontinuity will probably be large and the oscillations caused by it will be hard to damp out.
Chapter 5: Results and Evaluation

To illustrate the effectiveness of the interactive spline approximation, four test functions were chosen. The first three were taken from Franke who used these functions to test a large number of methods for interpolating scattered data. Some accuracy and timing results from his paper have been included. However, since the methods discussed in (Franke 79) are for scattered data, different data points had to be used for the generation of our spline surface and consequently, comparisons should be made cautiously as the methods have very different areas of application. The fourth test function is included to demonstrate the knot selection algorithm and the data insertion technique.

The domain of the three test functions from (Franke 79) is the unit square. To improve the scaling of the graphics, the sampled data was translated and scaled along the x and y axes to the square \([-1,1] \times [-1,1]\). The three test functions from Franke are:

Saddle function: \( f_1(X,Y) = \frac{1.25 + \cos(5.4Y)}{6(1 + (3x-1)^2)} \)

Exponential function:

\[
\begin{align*}
    f_2(X,Y) &= .75 \exp \left( -\frac{(9X-2)^2 + (9Y-2)^2}{4} \right) \\
    &\quad + .75 \exp \left( -\frac{(9X+1)^2/49 - (9Y+1)^2/10} \right) \\
    &\quad + .5 \exp \left( -\frac{(9X+7)^2 + (9Y-3)^2}{4} \right) \\
    &\quad - .2 \exp \left( -\frac{(9X-4)^2 - (9Y-7)^2} \right)
\end{align*}
\]

Cliff function: \( f_3(X,Y) = \frac{1}{9} (\tanh(9Y - 9X) + 1) \)

The test functions are plotted below. In Figures 7, 9 and 11 the X axis is horizontal, parallel to the page and the y axis is pointed away from the reader. In Figures 8 and 10 the picture has been
rotated 90 degrees counterclockwise around the z axis.

Figure 7: Function f1 (Saddle function)

Figure 8: Function f1 rotated 90 degrees
The second test function contains two hills and a depression:

```
SURFACE FITTING
1.04197599E+01
-1.85004711E+00
.939597893E+01
```

Figure 9 Function f2 (Exponential test function)

```
SURFACE FITTING
1.04197598E+01
-1.85004711E+00
.939597883E+01
```

Figure 10 Function f2 (Rotated 90 degrees)
The third test function contains a cliff running diagonally across the domain:

```
SURFACE FITTING
0.222383976E+00
-1.62141905E-03
0.372894764E+01
```

Figure 11 Function f3 (Cliff test function)
The accuracy of the spline fit for these examples is measured using 625 points on the domain of the function. The five worst errors \(|f(x_i,y_i) - s(x_i,y_i)|\) and their positions are given as well as an approximation to the least squares error. The spline surface for each example is generated using 2500 data points evenly spaced over the domain (unless otherwise specified).

Table I: Errors on function fl using evenly spaced knots

<table>
<thead>
<tr>
<th>KNOTS</th>
<th>ABS ERROR</th>
<th>L.S. ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 X 10</td>
<td>1.08E-3 at (-.265,-1 )</td>
<td>2.60E-4</td>
</tr>
<tr>
<td></td>
<td>1.06E-3 (-.265,-.837)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.05E-3 (-.265,-.918)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.935E-3 (-.265,-.755)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.852E-3 (-.265,-.673)</td>
<td></td>
</tr>
<tr>
<td>30 X 30</td>
<td>1.43E-5 (-.265,-1 )</td>
<td>3.46E-7</td>
</tr>
<tr>
<td></td>
<td>1.19E-5 (-.265,-.918)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.19E-5 (-.265,-.837)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.894E-5 (-.347,-1 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.835E-5 (-.265,.959 )</td>
<td></td>
</tr>
</tbody>
</table>
The second test function is more difficult to fit accurately and was used by Franke as his primary test function. This function is examined more extensively here as seen in the following results: The number of knots, the knot spacing and the number of data points are all varied.

Table II: Errors on function f2 using evenly spaced knots

<table>
<thead>
<tr>
<th>KNOTS</th>
<th>ABS ERROR</th>
<th>L.S. ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 X 10 (even)</td>
<td>2.85E-2</td>
<td>4.26E-3</td>
</tr>
<tr>
<td></td>
<td>2.59E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.89E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.87E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.85E-2</td>
<td></td>
</tr>
<tr>
<td>(225 data points)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 X 10 (even)</td>
<td>2.87E-2</td>
<td>4.43E-3</td>
</tr>
<tr>
<td></td>
<td>2.58E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.94E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.89E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.84E-2</td>
<td></td>
</tr>
<tr>
<td>(100 data points =&gt; interpolation)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 X 10 (even)</td>
<td>8.76E-2</td>
<td>1.11E-2</td>
</tr>
<tr>
<td></td>
<td>8.17E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7.30E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7.27E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.82E-2</td>
<td></td>
</tr>
<tr>
<td>(2500 data points)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30 X 30 (even)</td>
<td>8.39E-5</td>
<td>9.79E-6</td>
</tr>
<tr>
<td></td>
<td>6.70E-5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.37E-5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.16E-5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.08E-5</td>
<td></td>
</tr>
</tbody>
</table>
With a set of good knots, which were chosen using the package, the improved result given in Table III are obtained.

Table III: Errors on function f2 using a good knot placement

<table>
<thead>
<tr>
<th>KNOTS</th>
<th>ABS ERROR</th>
<th>L.S. ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2500 data points)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 X 10</td>
<td>1.38E-2</td>
<td>(.265,.551)</td>
</tr>
<tr>
<td></td>
<td>1.36E-2</td>
<td>(.265,.632)</td>
</tr>
<tr>
<td></td>
<td>1.21E-2</td>
<td>(-.510,-.510)</td>
</tr>
<tr>
<td></td>
<td>1.18E-2</td>
<td>(-.204,.469)</td>
</tr>
<tr>
<td></td>
<td>1.15E-2</td>
<td>(-.204,.551)</td>
</tr>
<tr>
<td>(225 data points)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 X 10</td>
<td>1.37E-2</td>
<td>(.265,.551)</td>
</tr>
<tr>
<td></td>
<td>1.34E-2</td>
<td>(.265,.632)</td>
</tr>
<tr>
<td></td>
<td>1.21E-2</td>
<td>(-.510,-.510)</td>
</tr>
<tr>
<td></td>
<td>1.21E-2</td>
<td>(-.204,.469)</td>
</tr>
<tr>
<td></td>
<td>1.19E-2</td>
<td>(-.204,.551)</td>
</tr>
</tbody>
</table>

Table IV lists some error results from function f4. Notice that the optimal knot spacing over this function is an even mesh and the optimal weighting is also an even distribution. It is not possible for the spline to accurately fit this function without spending additional time on areas that do not need the attention.

Table IV: Errors on function f3

<table>
<thead>
<tr>
<th>KNOTS</th>
<th>ABS ERROR</th>
<th>L.S. ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 X 10 (even)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.07E-3</td>
<td>(.224,.612)</td>
<td>1.84E-3</td>
</tr>
<tr>
<td>5.07E-3</td>
<td>(.612,.224)</td>
<td></td>
</tr>
<tr>
<td>4.84E-3</td>
<td>(.224,-.184)</td>
<td></td>
</tr>
<tr>
<td>4.84E-3</td>
<td>(-.184,.224)</td>
<td></td>
</tr>
<tr>
<td>4.72E-3</td>
<td>(.469,.612)</td>
<td></td>
</tr>
<tr>
<td>30 X 30 (even)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.95E-5</td>
<td>(.142,.224)</td>
<td>3.99E-6</td>
</tr>
<tr>
<td>1.91E-5</td>
<td>(-.673,-.591)</td>
<td></td>
</tr>
<tr>
<td>1.88E-5</td>
<td>(.633,.551)</td>
<td></td>
</tr>
<tr>
<td>1.87E-5</td>
<td>(.224,.142)</td>
<td></td>
</tr>
<tr>
<td>1.85E-5</td>
<td>(-.183,-.265)</td>
<td></td>
</tr>
</tbody>
</table>

Table V lists some representative results from (Franke 79). The
results presented were obtained by testing the method on 100 scattered data points.

Table V: Errors on function f2 by selected methods from (Franke 79)

<table>
<thead>
<tr>
<th>METHOD</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ABS</td>
<td>L.S.</td>
<td>ABS</td>
</tr>
<tr>
<td>Methods based upon points:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mod. Quad. Shepard</td>
<td>.0125</td>
<td>.00194</td>
<td>.0573</td>
</tr>
<tr>
<td>Shepard</td>
<td>-----</td>
<td>-----</td>
<td>.273</td>
</tr>
<tr>
<td>Methods based upon triangles:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lawson</td>
<td>.0565</td>
<td>.00359</td>
<td>.0951</td>
</tr>
<tr>
<td>Akima</td>
<td>.0274</td>
<td>.00423</td>
<td>.0647</td>
</tr>
<tr>
<td>Methods based upon squares:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Franke-TPS</td>
<td>.0165</td>
<td>.00273</td>
<td>.0940</td>
</tr>
<tr>
<td>Hardy</td>
<td>.00461</td>
<td>.00052</td>
<td>.0225</td>
</tr>
</tbody>
</table>

The modified quadratic Shepard's formula is basically equation (2.5) modified to have local support. Both Lawson's method and Akima's method use basis functions over a triangular grid. Franke's method uses local (Thin plate) functions over a rectangular mesh. Hardy's method is somewhat different from the methods previously discussed. It uses a local basis function (like the B-splines) but the support of the basis function is a circular region. A large sparse system of equations (like the one which could result from using B-splines for scattered data) must be solved to generate the surface.
In Table VI we present some timing results from the generation and evaluation of our spline surface. Table VII presents the timing results given in (Franke 79) for the methods presented in Table V. The times are in seconds. An Amdahl 470-V6-II was used to generate and evaluate the spline surfaces. An IBM 360-67 was used for all timings taken from (Franke 79).

Table VI: Timings from generation and evaluation of spline

<table>
<thead>
<tr>
<th>SPLINE SURFACE</th>
<th>PREPROCESSING (Generating the spline)</th>
<th>EVALUATION (625 points)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2500 data points)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(100 basis functions)</td>
<td>0.13</td>
<td>0.26</td>
</tr>
<tr>
<td>(900 basis functions)</td>
<td>0.19</td>
<td>0.26</td>
</tr>
<tr>
<td>(100 data points)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(100 basis functions)</td>
<td>0.05</td>
<td>0.26</td>
</tr>
</tbody>
</table>

Evaluation speed could be increased considerably since the algorithm used to generate the B-splines to evaluate the spline surface is for splines of arbitrary order and therefore quite slow.

Table VII: Timings on selected methods from (Franke 79)

<table>
<thead>
<tr>
<th>METHOD</th>
<th>PREPROCESSING</th>
<th>EVALUATION (1089 data points)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mod. Quad. Shepard</td>
<td>8.6</td>
<td>15</td>
</tr>
<tr>
<td>Shepard</td>
<td>---</td>
<td>17</td>
</tr>
<tr>
<td>Lawson</td>
<td>1.8</td>
<td>1.7</td>
</tr>
<tr>
<td>Akima</td>
<td>2.2</td>
<td>0.8</td>
</tr>
<tr>
<td>Franke-TPS</td>
<td>2.7</td>
<td>6.5</td>
</tr>
<tr>
<td>Hardy</td>
<td>7.1</td>
<td>13</td>
</tr>
</tbody>
</table>
Even allowing for the difference in speed between computers, the efficiency of the one dimensional spline approach to surface fitting is clearly demonstrated.

The fourth test function was included to show the adaptability of the spline using a good knot placement, and the ability of the spline to handle non-rectangular domains. The domain of the first example of function $f_4$ is the square $([-1,1] \times [-1,1])$. The function is given by:

$$f_4(x,y) = \frac{1}{1+25x^2} \cos \left( \frac{\pi}{2} y \right)$$

This function is one dimensional in nature. The surface varies the most along the $X$ axis near the origin.

Figure 12 Plot of function $f_4$
Figure 13 Function f4 rotated 90 degrees

The error results for a spline of even mesh over function f4 are given in Table VIII.

Table VIII: Errors from function f4 over an even mesh

<table>
<thead>
<tr>
<th>KNOTS</th>
<th>ABS ERROR</th>
<th>L.S. ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 X 13 (even)</td>
<td>1.33E-2 at (.143,-.204)</td>
<td>3.71E-3</td>
</tr>
<tr>
<td></td>
<td>1.32E-2 (.143,.612)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.31E-2 (.143,-.102)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.29E-2 (.143,.142)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.28E-2 (-.204,-.204)</td>
<td></td>
</tr>
</tbody>
</table>

A good knot sequence was chosen using routines based upon the one dimensional optimization technique presented by Dodson as described in Chapter 4.
Table IX: Spline approx. of function f4 over an optimized mesh

<table>
<thead>
<tr>
<th>KNOTS</th>
<th>ABS ERROR</th>
<th>L.S. ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 X 13 (opt)</td>
<td>2.73E-3</td>
<td>(.224,-.204)</td>
</tr>
<tr>
<td></td>
<td>2.71E-3</td>
<td>(.224,.612 )</td>
</tr>
<tr>
<td></td>
<td>2.68E-3</td>
<td>(.224,-.102)</td>
</tr>
<tr>
<td></td>
<td>2.65E-3</td>
<td>(.224,.143 )</td>
</tr>
<tr>
<td></td>
<td>2.62E-3</td>
<td>(.224,-.184)</td>
</tr>
</tbody>
</table>

The optimization routines will not produce a better knot sequence unless the fit is already quite good. In this case, 13 knots were needed along the X axis before the routines would work.

Table X presents the results of applying the techniques described in Chapter 4 to extend our spline method to non-rectangular domains. The domain of the function to be approximated is defined by the user supplied Fortran function "R1" (R1(x,y) is true if (x,y) is in the domain of f(x,y), otherwise it is false). For this example the domain of the function is L-shaped with ([-1,0] X [-1,0]) as the "hole" in the rectangular domain.

The surface varies very rapidly near (0,0). A more gentle surface would show much better results than this example. Error estimates are given after the initial patch and after each successive surface is generated. The maximum errors tend to move out of the "hole" as the weights are increased. Weights are applied to scan lines that "touch" the hole. As a result the weight on the data points in the hole is the product of the weight on the two lines that intersect at that data point.
Table X: Spline approx. of function f4 over an L shaped domain

<table>
<thead>
<tr>
<th>KNOTS</th>
<th>ABS ERROR</th>
<th>L.S. ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 X 13 (OPT) (PATCH WEIGHT 0.2)</td>
<td>1.16E-2 AT (-.204, -.428)</td>
<td>1.67E-3</td>
</tr>
<tr>
<td></td>
<td>1.14E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.11E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.06E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.01E-2</td>
<td></td>
</tr>
<tr>
<td>13 X 13 (opt) (SMOOTH WEIGHT .2)</td>
<td>5.53E-3</td>
<td>8.06E-4</td>
</tr>
<tr>
<td></td>
<td>3.17E-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.16E-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.06E-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.01E-3</td>
<td></td>
</tr>
<tr>
<td>13 X 13 (opt) (SMOOTH WEIGHT .5)</td>
<td>5.53E-3</td>
<td>7.69E-4</td>
</tr>
<tr>
<td></td>
<td>3.04E-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.91E-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.86E-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.86E-3</td>
<td></td>
</tr>
<tr>
<td>13 X 13 (opt) (SMOOTH WEIGHT 1 )</td>
<td>5.53E-3</td>
<td>7.64E-4</td>
</tr>
<tr>
<td></td>
<td>3.08E-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.85E-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.82E-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.78E-3</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 6: Conclusions

A method of generating a polynomial spline surface $s(x,y)$ approximating a function $f(x,y)$ represented by a set of regularly spaced data points over a "nearly" rectangular grid has been presented. The surface, although "global" in nature, can be generated efficiently for fairly large sets of data points. The point evaluation cost is low and can be reduced if the exact degree of the spline surface is known in advance (the B-splines can be evaluated more efficiently). The cost may be reduced still further at the expense of space if the representation of the surface is converted to a tensor product piecewise polynomial representation. As it is quite hard for the human eye to detect discontinuity in derivatives higher than 2 in a function, the surface approximant needs to be no more than $C^2$ to appear smooth. For lower degree fits (say $k = 4$ "cubic polynomials") the surface appears smooth and does not tend to oscillate as much as surfaces based upon higher degree polynomials.

When comparing this method to other approaches of surface fitting, it should be noted that other methods generally do not have all the restrictions this method has but very few are able to approximate the data (least squares or otherwise) rather than interpolate it. We have (perhaps unfairly) compared the method to methods for interpolating scattered data discussed in (Franke 79). The better methods presented there fit the test functions $f_1$, $f_2$, and $f_3$ in Chapter 5 well using one hundred data points. The spline fit does almost as well (in terms of accuracy) as the better methods presented on this amount of data with an even mesh and 100 basis functions (interpolation). Since a least-squares fit is used, it is
possible to increase the number of data points without increasing the number of basis functions. As the number of data points is increased the accuracy of the spline fit is also increased and becomes comparable with the best results in (Franke 79) while the computational cost is still very reasonable. The surface fitting package was used to carefully choose a good knot sequence over the function \( f_2 \). The accuracy of the surface using the carefully chosen knot sequence was considerably better than the fit generated using an even knot sequence as seen from the the results in Chapter 5. The cost of generating the spline surface and evaluation (even with a relatively large number of points) is nominal compared to the cost of generating and evaluating the more general surface fitting methods.

The restrictions imposed on the data and the domain imply that the data can be stored in \( \approx \frac{1}{3} \) the space it would take to store the \( (x_i, y_i, z_i) \) triplets. Similarly, the B-spline representation of the surface approximant is very compact; there are two vectors specifying the location of the surface patches (knots for the B-splines) and slightly more than one coefficient per patch is needed to specify the spline surface over these patches. In terms of space and speed this makes the spline fit very attractive relative to the more general surface fitting techniques. As discussed in Chapter 4, there are ways of extending the method to non-rectangular domains. If there are space or speed constraints on a problem or the data has errors in it then it may be justified to re-sample a problem defined with scattered data by using a more general surface fitting method and using the least squares spline fit over the re-sampled data.

The interactive environment supplied by the surface fitting
package provides a powerful tool to the user: A user can dynamically alter the fit of the spline surface until he is satisfied with the results (the graphics facilities allowing him to visualize and therefore judge the accuracy of the fit).

There probably is not much more work that can be done to improve the behavior of polynomial spline product surfaces. There may be some promise in surfaces generated using rational splines or non-polynomial functions but we believe the current research into this approach is well developed. There is considerable opportunity for research into more general two dimensional surface approximation techniques.

In conclusion the graphics package provided in this thesis effectively solves two dimensional surface fitting problems provided that the constraints on the data are met.
BIBLIOGRAPHY


Appendix A: A Surface Fitting Package

The code for the package is divided into two sections: The command interpreter and the support routines.

a) The Command Interpreter

The command interpreter is designed to provide an interactive environment which a user can fit a tensor product polynomial spline surface to his data. Its command structure is designed to protect the user as much as is practical while trying not to "get in the way". A description of the basic data structures and commands follow:

The Data Structures

There are basically four different "data objects" the user may manipulate in the package. These are:

1) Data: The data that specifies the surface consists of

   L   - A matrix of dimension at least LM x LN which contain
        the values of \( F_j \) at the grid intersection points.

   XX  - A vector of dimension at least LM containing the x
        coordinates.

   XY  - The corresponding vector in the y direction.

   WX  - A vector specifying the weights to apply to the grid lines
        specified in XX.

   WY  - The corresponding vector in the y direction.

   LM  - the number of grid lines perpendicular to the x axis.

   LN  - the number of grid lines perpendicular to the y axis.

   LMDIM - the first dimension of the matrix L.
2) Spline Surface: The components that specify the spline surface approximant.

GAMMA - A matrix of dimension at least M x N containing the coefficients of the B-spline surface approximant.

TX - The knot sequence specifying the B-splines in the x direction.

TY - The corresponding vector in the y direction.

KX - The order of the B-splines in the x direction.

KY - The order of the B-splines in the y direction.

M - The number of B-splines in the x direction.

N - The number of B-splines in the y direction.

MDIM - The first dimension of GAMMA.

3) Graphics: The internal 3-D graphical object (IMAGE)

X - A vector specifying the position of the grid lines perpendicular to the x axis.

Y - The corresponding y vector.

Z - The value of the surface at the points X @ Y

NX - The number of grid lines in x.

NY - The number of grid lines in y.

NZ - The first dimension of the matrix Z.

4) Graphics: The IG representation of the surface (DISPLAY)

(data available in surface package)

THETA - The angle the surface is viewed from (Azimuth).

PHI - The height (Elevation) surface is viewed from.
The system provides commands for generating, changing, initializing, reading and writing these objects. It also tries to keep the system consistent (e.g. a user may not generate the coefficients of a spline approximant before the data has been read in and the knots have been specified). Any non-ambiguous subset of a command may be used to specify that command (e.g INIT), and in most cases line boundaries are ignored when reading the commands. The commands available are:

**INITIALIZE**

Initialize mode allows the user to initialize the following variables:

**ORDER KX KY**

Initializes the order of the splines in the x and y direction. It invalidates the current spline surface if there is one.

**KNOTS M N**

Initializes the number of B-Splines to be used in the system. The knots are initialized to an even mesh with M B-splines in the x direction and N in the y. The knots are re-initialized every time the bounds may change (i.e. if new data is read in).

**DISPLAY NX NY**

Initialize the number of (evenly spaced) mesh lines in the display. The display lines are also initialized every time the bounds may change.
**REGION**

\[ \{ R_i \mid i = 1, 4 \} \]
\[ \emptyset \]

Indicate which user supplied function is to be used to specify the domain of the function to be approximated. If the domain is rectangular no function needs to be specified. If a region has been previously specified then invoking the command "REGION" with no region specified cancels the region (the knots specify a rectangular region implicitly).

**END**

To return to the top level.

**CHANGE**

This mode allows internal sections which allow the user to change the KNOTS, DATA, WEIGHTS and DISPLAY data structures interactively.

**ADD** \( K \)

Add \( K \) knots to the current knot sequence. The user chooses a knot by positioning the cross hairs (Tektronix) at the place desired on the knot sequence displayed on the screen and pressing anything but \(<\text{ENTER}>)\.

**DELETE** \( K \)

Delete \( K \) knots from the current knot sequence.

**OPTIMIZE**

This mode allows the user to obtain a good knot placement (for a one dimensional problem along a scan line) using the knot placement algorithm as discussed in Chapter 4.
**PICK**

Interactively choose the scan line the optimization technique is to be applied to.

**TRY** {(SAVE) \{ 0 \} }

Apply the optimization technique to the current scan line.
Save the suggested mesh if SAVE is specified.

**ROTATE**

Rotate the viewing direction 90 degrees (change the axis parallel to the screen).

**END**

Return to KNOT mode.

**ROTATE**

Rotate the viewing axis.

**END**

Return to CHANGE mode.

**DATA**

This mode facilitates the generation of data as described in Chapter 4. These commands do not invoke any graphics activities.

**PATCH**

Use a Coons' patch to patch the holes in the domain. A region must already have been specified if this command is invoked.
SMOOTH WEIGHT WX

Generate a spline approximant over the domain with weight WT on the scan lines that pass through holes in the domain. Fill in the holes with data generated using the spline.

END

Return to CHANGE mode.

WEIGHT

Weight mode allows the user to interactively change the weights applied to the (lines of data) over the surface.

WEIGHT WT

Specify the value of the weight to be inserted.

CHANGE K

Interactively change the weight to WT on K data lines.

ROTATE

Rotate to other axis.

END

Return to CHANGE mode.

DISPLAY

Interactively change the grid spacing on the internal graphical object (IMAGE). It only allows the user to change the grid if a spline surface is to be displayed as the data is discrete and cannot be sampled arbitrarily.

ADD K

Interactively add K lines on the display. This does not immediately affect the graphical object, screen or image.
DELETE K

Interactively delete K lines to the display.

REFRESH

Refresh the screen adding the new display lines.

ROTATE

Rotate the display on the screen adding the new display lines.

END

Return to CHANGE level.

END

Return to top level.

REFRESH/DISPLAY

Draw screen using current (IG) graphical object.

ROTATE

Rotate the display on the screen.

\[
\{ \text{COEFFICIENTS} \}
\]

\[
\{ \text{IMAGE K} \}
\]

READ

\[
\{ \text{KNOTS FROM filename} \}
\]

\[
\{ \text{DATA} \}
\]

Read the corresponding item from the file filename. The formats for the files are contained in the routines RDATA etc. in the I/O section of the subroutine library. For "IMAGE" K must be in (0 .. 3) where 0 means the current display (SCREEN) which is used to generate the IG graphical object.
WRITE

\[
\begin{align*}
\text{COEFFICIENTS} & \\
\text{IMAGE K} & \\
\text{KNOTS} & \\
\text{DATA} & \\
\text{SCREEN} & \{ \text{DEVICE= 'CALC'}, \text{ 'PRINT'} \} \\
\ & \{ g \}
\end{align*}
\]

Write the corresponding item onto the file filename.

GENERATE

\[
\begin{align*}
\text{COEFFICIENTS} & \\
\text{ERROR} & \\
\text{DISPLAY} & \{ \text{IMAGE FROM SPLINE SURFACE ERROR} \}
\end{align*}
\]

Generate the corresponding item.

MOVE

\[
\begin{align*}
\text{IMAGE K} & \text{ TO IMAGE L} \\
\text{SCREEN} & \text{ SCREEN}
\end{align*}
\]

Move the corresponding item (K in (1 .. 3)).

SUBTRACT

\[
\begin{align*}
\text{IMAGE K} & \text{ FROM IMAGE L} \\
\text{SCREEN} & \text{ SCREEN}
\end{align*}
\]

Subtract the corresponding units placing the result in the second entry.

END

Terminate the run.
b) The Support Routines

The support routines are divided into five main sections:

**GRAPHICS**

Routines that generate the graphical object, display it on the screen, read positions for display lines and knots off of the screen etc.

**I/O**

Routines that handle the input and output of DATA, KNOTS, COEFFICIENTS, & UNITS (Data that specifies enough information to generate a graphical object). These routines should be used to prepare the data and knots when using this package to generate a spline surface.

**OPTIMIZATION**

Routines that handle the one dimensional optimization.

**PATCHING**

Routines find the rectangles on the rectangle D' that enclose the regions not in the domain of the function to be approximated and generate the data to fill these "holes" using a Coons' patch.

**GENERATION AND EVALUATION**

The routines that generate the spline surface, evaluate it and provide some error estimates.
Appendix B : Generation and Evaluation

These two routines use the spline package in (de-Boor 77) to generate and evaluate a tensor product polynomial spline surface of arbitrary order.

SUBROUTINE LS COEF(GAMMA, TX, TY, KX, KY, M, N, NDIM, XX, XY, WX, WY, L, LM, LN, LMDIM, A, X)

CALCULATE THE COEFFICIENTS "GAMMA" FOR THE B-SPLINE SURFACE APPROXIMATION

GAMMA - COEFFICIENT MATRIX RETURNED.
TX - KNOT SEQUENCE ALONG X AXIS
TY - KNOT SEQUENCE ALONG Y AXIS
KX - ORDER OF SPLINE ALONG X AXIS
KY - ORDER OF SPLINE ALONG Y AXIS
M - NUMBER OF B-SPLINES IN X DIRECTION
N - NUMBER OF B-SPLINES IN Y DIRECTION
NDIM - SIZE OF GAMMA, AND ONE SIDE OF X (NDIM > MAX(M,N))
XX - X VALUES OF DATA POINTS.
XY - Y VALUES OF DATA POINTS.
WX - WEIGHTS ALONG X DIRECTION.
WY - WEIGHTS ALONG Y DIRECTION.
L - DATA MATRIX.
LM - NO. OF DATA POINTS ALONG X AXIS
LN - NO. OF DATA POINTS ALONG Y AXIS
LMDIM - FIRST DIMENSION OF DATA MATRIX
A - WORK SPACE FOR GENERATING SPLINE
X - WORK SPACE FOR GENERATING COEFFICIENT MATRIX.

REAL TX(1), TY(1), XX(1), XY(1), A(1), L(LMDIM, LN),
1 X(NDIM, LN), GAMMA(NDIM, NDIM), VNIKX(20), WX(1), WY(1)

ZERO MATRICES

MNMAX = M
IF ( N .GT. MNMAX) MNMAX = N
KMAX = KX
IF (KY .GT. KMAX) KMAX = KY
IMAX = KMAX * MNMAX
DO 10 I=1,IMAX
1 A(I) = 0.
10 CONTINUE
DO 20 I=1,M
  DO 20 J = 1, LN
    X(I,J) = 0.
  CONTINUE

C
ILEFT = KX
IMK = 0
DO 70 LI = 1, LM

C FIND INTERVAL "TX(ILEFT) <= XX(LI) < TX(ILEFT+1)"
30 IF (ILEFT .EQ. M) GO TO 40
IF (XX(LI) .LT. TX(ILEFT + 1)) GO TO 40
ILEFT = ILEFT + 1
IMK = ILEFT - KX
GO TO 30

C
PLACE VALUE OF ALL NON ZERO BSPLINES EVALUATED ON XX(L1)
IN VNIKX

40 CALL BSPLVN(TX, KX, 1, XX(L1), ILEFT, VNIKX)

C
GENERATE A L AND PLACE IN X

T
DO 60 JJ = 1, KX
  I = IMK + JJ
  WXBSPL = WX(L1) * VNIKX(JJ)
  DO 50 J = 1, LN
    X(I,J) = X(I,J) + WXBSPL * L(L1,J)
  CONTINUE
50 CONTINUE

T
GENERATE A A AND PLACE IN A

DO 60 M1 = JJ, KX
  J = IMK + M1
  IJK = IJ(J,I,KX)
  A(IJK) = A(IJK) + WXBSPL * VNIKX(M1)
60 CONTINUE
70 CONTINUE

T (J) T (J)
SOLVE A A X = A L TO OBTAIN THE MATRIX X

RATIO = 1.0E-10
EPS = 1.0E-77
NRHS = 1
ITER = 0
NSCALE = 0
DO 90 I = 1, LN
  CALL FBAND(A,X(I,I),M,KX,NRHS,RATIO,DET,JEXP,NSCALE,
    ITER,EPS)
NRHS = 2

CONTINUE

CLEAR MATRICES

DO 95 I=1,IMAX
   'A(I) = 0.
95 CONTINUE

DO 100 I = 1, MNMAX
   DO 100 J = 1, MNMAX
      GAMMA(I,J) = 0.
100 CONTINUE

ILEFT = KY
IMK = 0
DO 150 LI = 1, LN

FIND THE INTERVAL "TY(ILEFT) <= XY(L) < TY(ILEFT+1)"

IF (ILEFT .EQ. N) GO TO 120
IF (XY(L) .LT. TY(ILEFT +1)) GO TO 120
ILEFT = ILEFT + 1
IMK = ILEFT - KY
GO TO 110

CALL BSPLVN(TY, KY, 1, XY(L), ILEFT, VNIKX)

GENERATE B X AND PLACE IN GAMMA

DO 140 JJ = 1, KY
   I = IMK + JJ
   WYBSPL = WY(L) * VNIKX(JJ)
   DO 130 J = 1, M
      GAMMA(I,J) = GAMMA(I,J) + WYBSPL * X(J,L)
130 CONTINUE

GENERATE B B AND PLACE IN A

DO 140 Ml = JJ, KY
   J = IMK + Ml
   IJK = IJ(J,I,KY)
   A(IJK) = A(IJK) + WYBSPL * VNIKX(Ml)
140 CONTINUE

SOLVE B B GAMMA = B X TO OBTAIN GAMMA AND PLACE IN GAMMA

NRHS = 1
RATIO = 1.0E-10
DO 170 I = 1, M
   CALL FBAND(A,GAMMA(I,1),N,KY,NRHS,RATIO,DET,JEXP,NSCALE,
         1          ITER,EPS)
   NRHS = 2
   CONTINUE
   CALL COPY(GAMMA,TX,TY,KX,KY,M,N,NDIM)
RETURN
END

C
REAL FUNCTION SPL2D(X,Y)
C
THIS FUNCTION EVALUATES THE B-SPLINE FUNCTION DEFINED BY THE
COEFFICIENT MATRIX GAMMA AND THE KNOT SPQUENCES TX AND TY
C
COMMON /CA/ GAMMAT(30,30), TX(35), TY(35), M, NDIM, N, KX, KY
REAL BCOEF(20)
C
SPL2D(X,Y)=SUM (SUM GAMMA(I,J) B(Y)) B(X)
I J J,K,T I,H,S
C
CALL INTERV(TX, M + 1, X, LEFTX, MFLAG)
SPL2D1 = 0.
IF (MFLAG .NE. 0) GO TO 20
C
INNER LOOP FIRST
C
DO 10 J = 1, KX
   BCOEF(J) = BVALUE(TY,GAMMAT(1,LEFTX - KX + J),N,KY,Y,0)
10  CONTINUE
C
SPL2D = BVALUE(TX(LEFTX - KX + 1),BCOEF,KX,KX,X,0)
20  CONTINUE
RETURN
END