How to Approximate Scattered Data by Least Squares

Michael S. Floater
March 1998

Abstract: This report is designed to serve as a gentle guide to approximating scattered data by least squares methods. The spirit of the report is to keep the choice of basis functions arbitrary but we treat B-splines in detail because their special properties result in structured sparse matrices.

1. Introduction

This is a collection of notes I compiled while in the process of implementing a least squares method for approximating bivariate scattered data by tensor-product splines. This method seems to yield very well behaved, smooth surfaces even when the data set is very large and is certainly an attractive alternative to other scattered data methods such as piecewise polynomials over triangulations or radial basis functions.

Writing these notes helped me to clarify my ideas during the implementation, which is not as trivial as one might first imagine from the mathematics! I hope that reading these notes will help others in the future.

I have drawn on several useful sources, the main one being the paper by von Golitschek and Schumaker [6] which covers quite a lot of the material here. Several others have implemented least squares methods in various contexts [1,7,9,10].

2. Univariate Data

We begin by considering univariate *scattered data*, a sequence of distinct real values $x_1, \ldots, x_m$ (not necessarily ordered) together with arbitrary real values $z_1, \ldots, z_m$. We wish to approximate this data by a function

$$s(x) = \sum_{i=1}^{n} B_i(x) c_i, \qquad c_i \in \mathbb{R}.$$

Loosely speaking we say that a function s *approximates* the data if $s(x_k) \approx z_k$. The functions $B_1, \ldots, B_n$ are defined over some interval $[a, b]$ and could for example be B-splines, Bernstein polynomials, monomials, radial functions etc.

The basic *least squares* method consists of finding a coefficient vector $c = (c_1, \ldots, c_n)^T$ which minimizes the sum of squared errors

$$\sum_{k=1}^{m} (s(x_k) - z_k)^2. \tag{2.1}$$

Clearly if the minimum is zero then the function s *interpolates* the data, i.e., $s(x_k) = z_k$.

This basic minimization may be adequate in some practical situations but there are several good reasons to add to the sum a smoothing term such as

$$J(c) = \int_a^b (s''(x))^2 \, dx \tag{2.2}$$

(assuming the functions $B_1, \ldots, B_n$ are C^2). One reason is that the data may be noisy; another is that the minimum of the sum in (2.1) may not be unique. There are many possible smoothing terms used in practice but usually they can be expressed as

$$J(c) = c^T E c$$

for some $n \times n$ symmetric positive semidefinite matrix E. For example the smoothing term in (2.2) can be rewritten as

$$J(c) = \int_a^b \Big(\sum_{i=1}^n B_i''(x) c_i\Big)\Big(\sum_{j=1}^n B_j''(x) c_j\Big) dx = \sum_{i=1}^n \sum_{j=1}^n E_{ij} c_i c_j$$

where

$$E_{ij} = \int_a^b B_i''(x) B_j''(x) dx. \tag{2.3}$$

So we will concentrate on minimizing the more general functional

$$F(c) = \sum_{k=1}^m (s(x_k) - z_k)^2 + \lambda c^T E c \tag{2.4}$$

for some real $\lambda > 0$.

A minimum of $F(c)$ must occur at a point c where all partial derivatives are zero, a *critical point*. The equations $\partial F / \partial c_i = 0$ are called the *normal equations* of the least squares problem. By differentiating F in (2.4) explicitly and rearranging the subsequent expression, the normal equations can be rewritten as the single matrix equation

$$(B^T B + \lambda E) c = B^T z, \tag{2.5}$$

where $z = (z_1, \ldots, z_n)^T$ and B is the $m \times n$ matrix

$$B = (B_j(x_i))_{i=1,\ldots,m, j=1,\ldots,n} = \begin{pmatrix} B_1(x_1) & \ldots & B_n(x_1) \\ \vdots & & \vdots \\ B_1(x_m) & \ldots & B_n(x_m) \end{pmatrix}.$$

Then the solution to the least squares problem is the solution c to (2.5). As observed in [6], the $n \times n$ matrix $G = B^T B$, whose ij-th element is

$$G_{ij} = \sum_{k=1}^m B_i(x_k) B_j(x_k),$$

is symmetric and positive semidefinite. Moreover if $m \geq n$ and the matrix B has full rank n (n of its rows are linearly independent), the matrix G is positive definite. In this case, equation (2.5) has a unique solution. However even if G is not positive definite, if E is positive definite then so is $G + \lambda E$ and there is still a unique solution.

3. B-splines

Let us now suppose that the functions $B_1, \ldots, B_n$ in (2.1) are B-splines in which case the function s in (2.1) is a spline. It pays to treat this case separately since the matrix B in (2.5) is then sparse.

We write $B_{i,K} = B_i$ to indicate that the B-splines have order K and they are defined over some knot sequence

$$t_1, t_2, \ldots, t_{n+K}.$$

We make the usual assumptions that $t_i \leq t_{i+1}$ and $t_i < t_{i+K}$. One way of defining the B-splines is recursively by the Cox-de Boor algorithm. We define

$$B_{i,1}(x) = \begin{cases} 1 & t_i \leq x < t_{i+1}; \\ 0 & \text{otherwise}, \end{cases}$$

and, for $K \geq 2$,

$$B_{i,K}(x) = \frac{x - t_i}{t_{i+K-1} - t_i} B_{i,K-1}(x) + \frac{t_{i+K} - x}{t_{i+K} - t_{i+1}} B_{i+1,K-1}(x). \tag{3.1}$$

The piecewise polynomial $B_{i,K}$ has support $[t_i, t_{i+K}]$ and is (strictly) positive in the open interval (t_i, t_{i+K}). Though the B-splines are defined over the whole of the real line, a natural choice of the domain $[a, b]$ over which the smoothing integral in (2.2) can be taken is the one defined by

$$a = t_K, \qquad b = t_{N+1}$$

because for $x \in [t_K, t_{N+1}]$,

$$\sum_{i=1}^{n} B_{i,K}(x) = 1.$$

There are two important issues concerning the computational speed of finding the solution to the linear system (2.5) in this case. One is related to the size of n, the number of B-splines B_i, and the other to the size of m, the number of scattered data points (x_k, z_k).

3.1. Solving the linear system

The size of the matrix

$$A = G + \lambda E \tag{3.2}$$

is $n \times n$ so we should take care about the structure of A when n is large. In fact assuming that the smoothing term defining E is that in (2.2), A is *sparse*, something which should be exploited by the linear system solver. The point is that the product $B_i B_j$ of two B-splines is zero if the interiors of their supports are disjoint, which occurs when $|j - i| \geq K$. Therefore all elements G_{ij} of G are zero outside a diagonal band of width $2K-1$. Similarly the elements E_{ij} in (2.3) are also zero outside this band and so also A is banded with width $2K-1$. This fact should be exploited by the equation solver. One could for example write a tailored Gauss elimination or a Cholesky decomposition.

Let us note that even elements of the band in G may also be zero, depending on the locations of the scattered data. Indeed there may be indices i, j such that $|j - i| < K$

and at the same time there are no values x_k belonging to the non-empty intersection $(t_i, t_{i+K}) \cap (t_j, t_{j+K})$. In this case $G_{ij} = 0$. However it is unlikely that the contribution E_{ij} from the smoothing term to $A_{ij} + \lambda E_{ij}$ is also zero and so it does not seem worthwhile taking into account further possible zeros within the band of A.

3.2. Constructing the matrix A

The dimension of the matrix A in (3.2), $n \times n$, is clearly independent of the number of scattered data m and so the computational time required to *solve* equation (2.5) is also independent of m. However m plays a role in the *construction* of A, a role which could be critical when m is large. One of the main purposes behind approximating by least squares in the first place is to be able to approximate very large data sets. If m, for example, is of the order of 10^6 while n is of the order of 10, it could take considerably more CPU time to compute the elements A_{ij} than to solve equation (2.5). The bottleneck in the computation of A is constructing G.

The obvious way of computing the elements of G is to compute each G_{ij} in turn. But this could be terribly costly as not only does each of these computations require looping through all m scattered data, but the tree-structured recursive algorithm (3.1) for the B-splines will be used unnecessarily many times at the same value x_k. The simple trick is to instead process each x_k in turn and compute *all* the B-splines whose supports contain it, applying the Cox de Boor algorithm (3.1) just once. Specifically, if $t_i \leq x_k < t_{i+1}$, the only relevant B-splines are $B_{i-K+1}, \ldots, B_i$. Then each non-zero product $B_i(x_k)B_j(x_k)$ is added to the current value of G_{ij}.

4. Bivariate Data

One can easily generalize the least squares method of Section 2 to a multivariate (d-variate) setting by replacing the distinct values $x_1, \ldots, x_m$ by distinct points $\mathbf{x}_1, \ldots, \mathbf{x}_m$ in $\mathbb{R}^d$. We search for a d-variate approximant of the form

$$s(\mathbf{x}) = \sum_{i=1}^{n} B_i(\mathbf{x}) c_i,$$

to the scattered data $(\mathbf{x}_1, z_1), \ldots, (\mathbf{x}_m, z_m)$ by minimizing a functional of the form (2.4), replacing x_k by $\mathbf{x}_k$.

Yet our main concern is approximation of bivariate data by tensor-product functions and the special nature of such functions warrants a better way of indexing. So we change our focus now to approximation of scattered data $(x_1, y_1, z_1), \ldots, (x_m, y_m, z_m)$ (where the points $\mathbf{x}_i = (x_i, y_i)$ are taken to be distinct) by a function

$$s(x, y) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} B_{ij}(x, y) c_{ij}, \tag{4.1}$$

where B_{ij} is the tensor-product

$$B_{ij} = B_i \otimes C_j$$

defined by

$$(B_i \otimes C_j)(x, y) := B_i(x)C_j(y),$$

and $B_1, \ldots, B_{n_1}$ are functions defined on an interval $[a_1, b_1]$ and $C_1, \ldots, C_{n_2}$ on $[a_2, b_2]$.

In order to introduce vector and matrix notation we need to choose an ordering of the coefficients, so let us define

$$c = (c_{1,1}, \ldots, c_{n_1,1}, c_{1,2}, \ldots, c_{n_1,n_2})^T.$$

Then the task is to find the coefficient vector c, of length $n = n_1 n_2$, which minimizes

$$F(c) = \sum_{k=1}^{m} (s(x_k, y_k) - z_k)^2 + \lambda c^T E c,$$

where E is $n \times n$. Similar to the univariate case, a minimum of F occurs when c is the solution of the normal equations (2.5) where now B is the $m \times n$ matrix

$$B = \begin{pmatrix} B_1(x_1)C_1(y_1) & B_2(x_1)C_1(y_1) & \ldots & B_{n_1}(x_1)C_{n_2}(y_1) \\ \vdots & \vdots & & \vdots \\ B_1(x_m)C_1(y_m) & B_2(x_m)C_1(y_m) & \ldots & B_{n_1}(x_m)C_{n_2}(y_m) \end{pmatrix},$$

so

$$G_{(j-1)n_1+i,(s-1)n_1+r} = \sum_{k=1}^{m} B_i(x_k)C_j(y_k)B_r(x_k)C_s(y_k)$$

for $i, r = 1, \ldots, n_1$ and $j, s = 1, \ldots, n_2$.

As the smoothing term we could take the thin plate spline energy

$$J(c) = \int_{a_1}^{b_1} \int_{a_2}^{b_2} (s_{xx}^2 + 2s_{xy}^2 + s_{yy}^2) dy\, dx \tag{4.2}$$

which, after substituting in the definition of s in (4.1), can be expressed as

$$\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \sum_{r=1}^{n_1} \sum_{s=1}^{n_2} E_{ijrs} c_{ij} c_{rs},$$

where

$$E_{ijrs} = A_{ijrs} + 2B_{ijrs} + C_{ijrs}$$

and

$$A_{ijrs} = \int_{a_1}^{b_1} B_i''(x)B_r''(x)dx \int_{a_2}^{b_2} C_j(y)C_s(y)dy,$$

$$B_{ijrs} = \int_{a_1}^{b_1} B_i'(x)B_r'(x)dx \int_{a_2}^{b_2} C_j'(y)C_s'(y)dy,$$

$$C_{ijrs} = \int_{a_1}^{b_1} B_i(x)B_r(x)dx \int_{a_2}^{b_2} C_j''(y)C_s''(y)dy.$$

If the functions B_i and C_j are B-splines of orders K and L respectively then as in the univariate case, the matrices G and E are sparse and therefore A is sparse. In fact in this case

$$A_{(j-1)n_1+i,(s-1)n_1+r} = 0$$

if either $|i-r| \geq K$ or $|j-s| \geq L$. The non-zero elements of A are shown in Figures 1, 2, 3 for different values of n_1 and n_2 when the spline $s(x,y)$ is bicubic ($K = L = 4$). It is interesting to see how the structure of A can be viewed as "bands within a band". The size of all the small bands is n_1 while the size of the single large one is n_2. The matrix in each case has dimension $n_1n_2 \times n_1n_2$, in particular 100×100 in Figure 1 and 200×200 in Figures 2 and 3.

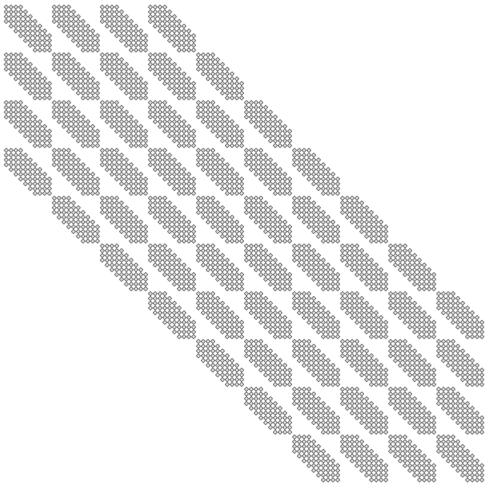

Fig. 1. Structure of matrix A when $K = L = 4$ and $n_1 = n_2 = 10$.

There are several known approaches to solve the basic least squares problem, i.e. minimizing (2.1), such as Cholesky decomposition of $G = B^TB$ in the normal equations or methods such as QR factorization applied directly to the minimization of $||Bc-z||^2$ (avoiding the normal equations altogether). However since we are applying a smoothing term, a different approach seems warranted. We could again carry out a Cholesky decomposition, this time on the matrix $A = B^TB + \lambda E$. But the band width is of order n_1. We have chosen instead to use the conjugate gradient method which capitalizes on the sparse form of A. For B-spline orders of K and L the number of non-zero elements of A in any row is at most $(2K-1)(2L-1)$ and is therefore independent of n_1 and n_2. For example, in Figures 1, 2, 3, the maximum number of non-zeros per row is 49.

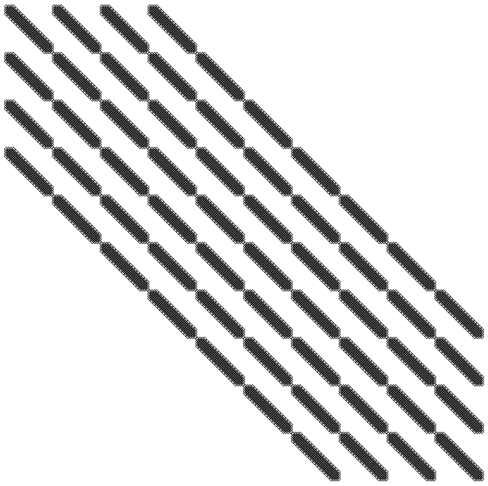

Fig. 2. Structure of matrix A when $K = L = 4$ and $n_1 = 20$, $n_2 = 10$.

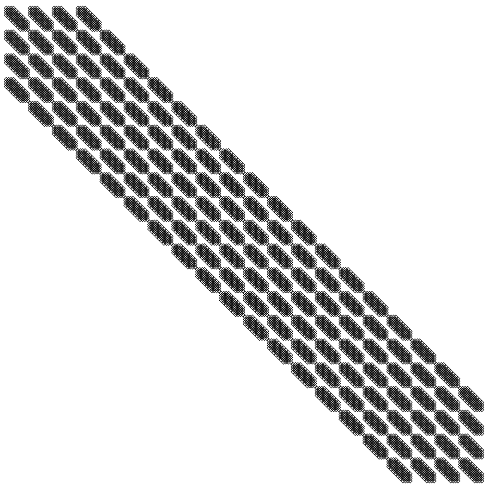

Fig. 3. Structure of matrix A when $K = L = 4$ and $n_1 = 10$, $n_2 = 20$.

5. Parametric surfaces

If we replace the function s in (4.1) by a parametric surface $\mathbf{s} : D \to \mathbb{R}^3$ by also replacing the coefficients c_{ij} by points $\mathbf{c}_{ij}$ in $\mathbb{R}^3$ then there are several possibilities for least squares approximation. We wish now to minimize the errors $||\mathbf{s}(x_k, y_k) - \mathbf{z}_k)||$ where $\mathbf{z}_1, \ldots, \mathbf{z}_m$ are given points in $\mathbb{R}^3$ and the points (x_k, y_k) in $\mathbb{R}^2$ are *parameter points* associated with the $\mathbf{z}_k$. If the given points $\mathbf{z}_k$ are structured, for example they are the vertices of some given triangulation, then a method for determining good parameter points (x_k, y_k) is described in [2]. If more generally they are structured in a tiling the method of [2] can be easily generalized [3].

There are several smoothing terms one can add to the squared errors, such as ones defined in terms of curvature, torsion, etc. Several approaches, linear and nonlinear, are mentioned in Dietz [1] and Greiner and Hormann [7]. We have found in practice that provided the parameter points are well chosen, good surface approximations result from simply applying the thin-plate spline functional to each of the three coordinates. This results in three independent sets of (linear) normal equations in which the matrix is the same, only the right hand sides differ. Specifically we propose minimizing

$$
\begin{aligned}
F(\mathbf{c}) &= \sum_{k=1}^{m} \|\mathbf{s}(x_k, y_k) - \mathbf{z}_k\|^2 + \lambda\left((c^1)^T E c^1 + (c^2)^T E c^2 + (c^3)^T E c^3\right) \\
&= \sum_{j=1}^{3} \left(\sum_{k=1}^{m} (s^j(x_k, y_k) - z_k^j)^2 + \lambda (c^j)^T E c^j \right),
\end{aligned}
\tag{5.1}
$$

where, as previously, E is the matrix for which $J(c) = c^T E c$ and $J(c)$ is as in (4.2). The solution vectors c^1, c^2, c^3 to this problem are the solutions respectively to the three normal equations

$$
(B^T B + \lambda E) c^j = B^T z^j, \qquad j = 1, 2, 3. \tag{5.2}
$$

The thought now arises as to whether, instead of solving all three equations iteratively and independently, it might be worth finding the inverse of the matrix $A = B^T B + \lambda E$ once and then simply computing $A^{-1} B^T z^j$ for $j = 1, 2, 3$. But this will only pay off if the time taken to invert A takes no more than a third of the time taken to solve each equation in (5.2). When the dimension, $n_1 n_2$, of A is large this alternative approach is therefore unlikely to succeed.

6. Final Remarks

6.1. Data thinning

For some data sets the number of points m is so large that computing the elements of A takes unreasonable CPU time (though as mentioned earlier the dimensions of the matrix A in the normal equations is independent of m). If m is for example of the order of 100,000 or more it may be acceptable to simply throw away several of the data points. This will depend of course on the density of the sample and the amount of detail in the object being measured. One approach to reducing the data set in an intelligent way is to employ the thinning-algorithm described in [4]. In that algorithm, points are removed recursively one by one. At each iteration, the 'least important point' in the current subset of points is removed. A dynamic triangulation is used to reduce computational cost to $O(m^2)$ (a naive algorithm would require $O(m^3)$). Moreover, we describe in a recent article [5] how the use of a heap data structure to implement a priority queue reduces the number of operations required to thin the data set to $O(m \log m)$. The heap algorithm has been used to thin data sets originally containing $\approx 300,000$ points down to $10,000$.

6.2. Choosing λ

Von Golitschek and Schumaker [6] described a method for estimating a value of λ, the scaling parameter in (2.4) of the smoothing term. The choice they propose has some nice theoretical properties under certain assumptions on the noise of the measurements, but seems to be very costly to compute (even one step of the cross-validation procedure is much more costly than solving the normal equations in the first place). I therefore chose to take a very pragmatic approach to estimating λ in the absence of anything better. In my implementation I have simply set

$$\lambda = ||B^T B|| / ||E||,$$

where the matrix norm $||A||$ of a matrix A is taken to be the l_2 norm $\sqrt{(\sum_{ij} a_{ij}^2)}$. The effect of this choice of λ is roughly speaking to ensure that the two contributions $B^T B$ and λE to the matrix $A = B^T B + \lambda E$ have equal weight. This choice has performed very well in the examples I have computed.

6.3. Parameter correction

Once the parametric surface $\mathbf{s}$ has been computed, one can find new parameter points (x'_k, y'_k) by minimizing for each k, the distance $||\mathbf{s}(x'_k, y'_k) - \mathbf{z}_k||$ of the data point $\mathbf{z}_k$ from the surface $\mathbf{s}$ (because the point $\mathbf{s}(x_k, y_k)$ may not be the closest point on $\mathbf{s}$ to $\mathbf{z}_k$). This procedure is known as parameter correction [8] and can be used to reduce the error of approximation as well as reduce oscillations due to a skew parametrization.

6.3. Constraints

In some applications it may be desirable to minimize the functional $F(c)$ subject to some constraints. The kind of constraint we are perhaps most likely to encounter in geometric modelling is a boundary condition. Typically we require that one or more layers of coefficients (or points) near the boundary of the grid $(\mathbf{c}_{ij})_{i=1,\ldots,n_1, j=1,\ldots,n_2}$ are fixed. This mean that the surface $\mathbf{s}$ can be constrained to interpolate a set of four spline boundary curves. With this kind of constraint, the normal equations just need to be modified accordingly.

6.4. A numerical example

A data set of approximately 300,000 points sampled from a model of a human face was triangulated and thinned down to 10,000 points, shown in Figure 1, using the thinning algorithm described in [5]. The new triangulation of 10,000 vertices $\mathbf{z}_k$ was subsequently mapped into the unit square in the plane using the shape-preserving parametrization of [2]. This provides parameter points (x_k, y_k).

The least squares method of Section 5 was then applied to the parametrized points. The spline surface was chosen to be bicubic, i.e. $K = L = 4$, and to have uniform knot vectors with $n_1 = n_2 = 100$. The resulting surface is shown shaded in Figure 2 and via isocurves in Figure 3. The conjugate gradient method was used to solve the three linear systems.

Acknowledgement. I wish to thank Øystein Gomo of Lærdal Medical for kind permission to use their 'face' data set in this report.

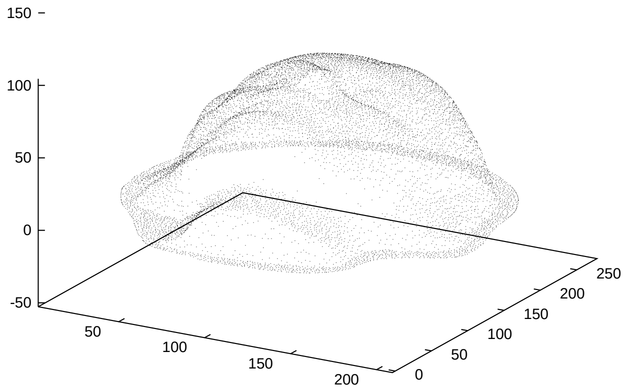

Fig. 1. Data set of $m = 10,000$ points.

Fig. 2. Approximate surface $\mathbf{s}(x, y)$.

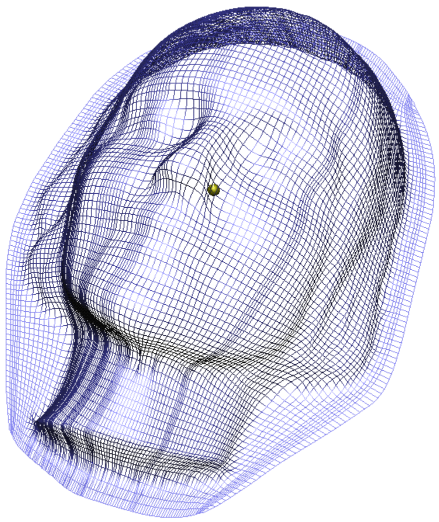

Fig. 3. Isocurves of the surface $\mathbf{s}(x, y)$.

References

1. U. Dietz, B-spline approximation with energy constraints, in *Advanced Course on Fairshape*, J. Hoschek and P. Kaklis (eds.), Teubner, Stuttgart, 1996, 229–240.
2. M. S. Floater, Parametrization and smooth approximation of surface triangulations, Comp. Aided Geom. Design **14** (1997), 231–250.
3. M. S. Floater, Parametric tilings and scattered data approximation (preprint), SINTEF Report No. STF42 A97012, Oslo, (1997).
4. M. S. Floater and A. Iske, Multistep Scattered Data Interpolation using Compactly Supported Radial Basis Functions, J. Comp. Appl. Math. **73** (1996), 65–78.
5. M. S. Floater and A. Iske, Thinning algorithms for scattered data interpolation, to appear in BIT, 1998.
6. M. von Golitschek and L. L. Schumaker, Data fitting by penalized least squares, in *Algorithms for Approximation II*, J. C. Mason and M. G. Cox (eds.), Chapman & Hall (London), 1990, 210-227.
7. G. Greiner and K. Hormann, Interpolating and approximating scattered 3D data with hierarchical tensor product B-splines, in *Surface Fitting and Multiresolution Methods*, A. Le Méhauté, C. Rabut, and L. L. Schumaker (eds.), Vanderbilt University Press, Nashville 1997, 163–172.
8. J. Hoschek and D. Lasser, *Fundamentals of Computer Aided Geometric Design*, AKPeters, Wellesley, 1994.
9. C. L. Lawson and R. J. Hanson, Solving Least Squares Problems, SIAM, Philadelphia, 1995.
10. V. Skytt, Geometric Manipulation of True Features: Surface Editing, SINTEF Report No. STF33 F93040, 1993.