Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computational Statistics & Data Analysis 50 (2006) 61 – 76
COMPUTATIONAL STATISTICS & DATA ANALYSIS
www.elsevier.com/locate/csda

Fast and compact smoothing on large multidimensional grids

Paul H.C. Eilers[a,*], Iain D. Currie[b], Maria Durbán[c]

[a]Department of Medical Statistics, Leiden University Medical Centre, P.O. Box 9604, 2300 RC Leiden, The Netherlands
[b]Heriot-Watt University, Edinburgh, Scotland
[c]Universidad Carlos III de Madrid, Madrid, Spain

Available online 20 August 2004

Abstract

A framework of penalized generalized linear models and tensor products of B-splines with roughness penalties allows effective smoothing of data in multidimensional arrays. A straightforward application of the penalized Fisher scoring algorithm quickly runs into storage and computational difficulties. A novel algorithm takes advantage of the special structure of both the data as an array and the model matrix as a tensor product; the algorithm is fast, uses only a moderate amount of memory and works for any number of dimensions. Examples are given of how the method is used to smooth life tables and image data.
© 2004 Elsevier B.V. All rights reserved.

Keywords: B-splines; Difference penalty; Multidimensional array; P-splines; Smoothing; Tensor product

1. Introduction

Many types of data come as large grids of values. Images are an obvious example, as are multidimensional (optical) spectra and histograms. In many cases smoothing is needed, to reduce noise, to estimate a trend, or to interpolate missing data. In one dimension, smoothing has essentially been solved, whether the data are on a grid or not: fast and statistically sound algorithms have been developed for this purpose and are available in software packages and

* Corresponding author
E-mail address: p.eilers@lumc.nl (P.H.C. Eilers).

0167-9473/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2004.07.008

toolboxes. For multidimensional grids the situation is not so rosy. Unless the data set is relatively small (a few thousand grid cells) memory consumption and computation quickly get out of hand. Many images and other multidimensional data contain a million or more cells and so severely challenge most algorithms.

The purpose of this paper is to present extensions of the P-spline approach (Eilers and Marx, 1996) to multidimensional grids. The smooth multidimensional surface is constructed from the tensor products of B-splines. The width of the B-splines is chosen to be relatively small and so a relatively large number of tensor products form the basis functions. This will generally overfit the data, and so, to increase smoothness further, difference penalties on neighbouring coefficients of the tensor products are introduced. The weights of these penalties give continuous control over smoothness, and these weights can then be tuned to the data.

A straightforward implementation of this approach would vectorize the data on the grid, flatten the tensor products to a regression basis, compute the normal equations, add penalties and solve the system. The size of this system is not a big problem, assuming that the total number of tensor products is of the order of 1000 or less, but the intermediate step with the flattened basis can lead to enormous problems. Imagine an image of 1000 by 1000 pixels and 1000 tensor products: the basis matrix would have 10^9 elements and thus take up 8 Gb of memory! In low-level languages like C or FORTRAN we can avoid this step by writing clever loops to compute the normal equations directly from the data. We prefer to work in a high-level language like Matlab or S-PLUS/R. Our algorithm avoids computation of the full basis matrix and computes the normal equations directly but without using C- or FORTRAN-like loops.

It turns out that, by rearranging the computations, we can work with matrices of the same size as the data matrix, thereby achieving enormous gains in both computational speed and memory use. We explain the algorithm and provide some applications. As a preparation, we summarize one-dimensional P-spline smoothing (Eilers and Marx, 1996) in the next section. Then we generalize P-splines to arbitrary data in two dimensions using tensor products; we discuss the problems in computational time and memory use that may arise. In Section 4 we show how, by rearranging the evaluation of sums of products, we can alleviate these problems, if the data are on a grid. In Section 5 we introduce a formalism for multiplication of matrices and arrays of any dimension; using this tool we generalize efficient smoothing on a grid to arbitrary dimensions. The next section presents applications to simulated and real-life data. A short discussion closes the paper.

2. P-spline smoothing in one dimension

B-splines are local basis functions, consisting of polynomial segments of low degree, commonly quadratic or cubic. The positions where the segments join are called the knots. B-splines have local support and are thus suitable for smoothing and interpolating data with complex patterns. They also reduce smoothing to linear regression, with large advantages when one needs standard errors, builds semiparametric models or works with non-normal data. Unfortunately, control over smoothness is limited: one can only change the number and positions of the knots. If there are no reasons to assume that smoothness is non-uniform, the knots will be equally spaced and the only tuning parameter is their (discrete) number. B-splines also have difficulties with sparse data: some of the basis functions may have little or no support and so their coefficients may be unstable or even impossible to estimate.

Eilers and Marx (1996) proposed the P-spline recipe: (1) use a (quadratic or cubic) B-spline basis with a large number of knots, say 10–50; (2) introduce a penalty on (second or third order) differences of the B-spline coefficients; (3) minimize the resulting penalized likelihood function; (4) tune smoothness with the weight of the penalty, using cross-validation or AIC to determine the optimal weight.

Let y and x, each vectors of length m, represent the observed and explanatory variables, respectively. Once a set of knots is chosen, the B-spline basis B follows from x. If there are n basis functions then B is $m \times n$. In the case of normally distributed observations the model is $y = B\alpha + e$, with independent errors e. Non-normal data will be discussed below. In the case of B-spline regression the sum of squares of residuals $\|y - B\alpha\|^2$ is minimized and the normal equations $B'B\hat{\alpha} = B'y$ are obtained; the explicit solution $\hat{\alpha} = (B'B)^{-1}B'y$ results. The P-spline approach minimizes the penalized least-squares function

$$S = \|y - B\alpha\|^2 + \lambda \|D_d \alpha\|^2, \tag{1}$$

where D_d is a matrix that forms differences of order d, i.e. $D_d\alpha = \Delta^d \alpha$. Examples of this matrix, for $d = 1$ and $d = 2$ are:

$$D_1 = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}; \quad D_2 = \begin{bmatrix} 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \end{bmatrix}. \tag{2}$$

Minimizing S leads to the penalized normal equations

$$(B'B + \lambda D_d' D_d)\hat{\alpha} = B'y, \tag{3}$$

from which an explicit solution for $\hat{\alpha}$ follows immediately. An important point is that the size of the system of equations is equal to the number of B-spline functions in the basis, independent of the number of observations.

The positive parameter λ determines the influence of the penalty. If λ is zero, we are back to B-spline regression; increasing λ makes $\hat{\alpha}$, and hence $\hat{\mu} = B\hat{\alpha}$, smoother. Eilers and Marx (1996) show that, as $\lambda \to \infty$, $\hat{\mu}$ approaches the best fitting (in the least squares sense) polynomial of degree $d - 1$.

From (3) it follows that

$$\hat{y} = B(B'B + \lambda D_d' D_d')^{-1} B'y = Hy. \tag{4}$$

The matrix H is commonly called the hat matrix; it is an extremely useful tool. It first shows that, for a given λ, the smoother is linear. Further, the trace of H gives a measure of the effective dimension of the model (Hastie and Tibshirani, 1990, p. 52):

$$ED = \mathrm{tr}(H) = \mathrm{tr}(B(B'B + \lambda D_d' D_d)^{-1} B') = \mathrm{tr}((B'B + \lambda D_d' D_d)^{-1} B'B). \tag{5}$$

ED ranges from n, the size of the B-spline basis, when λ=0, to d, the order of the differences, when λ becomes very large. Note that H is not a projection matrix, i.e., $H^2 \neq H$, as in standard linear regression: smoothing $\hat{y}$ again does not reproduce $\hat{y}$.

A common way to optimize a smoothing parameter is leave-one-out cross-validation. Imagine leaving observation i out of the data, fitting the model and computing an interpolated value $\hat{y}_{-i}$ and the deletion residual $y_i - \hat{y}_{-i}$. Repeating this for all observations, $i = 1, \ldots, m$, we arrive at the cross-validation sum of squares $CV = \sum_i (y_i - \hat{y}_{-i})^2$ (Hastie and Tibshirani, 1990, p. 43). If followed literally, this would be a rather expensive recipe, certainly when we have many observations. However we can show that

$$y_i - \hat{y}_{-i} = \frac{y_i - \hat{y}_i}{1 - h_{ii}}, \quad i = 1, \ldots, m, \tag{6}$$

so, if we know the diagonal of H, we can compute deletion residuals very efficiently. For large data sets it is advisable to arrange the computations is such a way that only the diagonal of H is computed. Thus, with proper organization, we can compute CV with little extra work.

Weights can be introduced straightforwardly, to account for possible unequal variance of the observations. Weights can also be used for interpolation and extrapolation. If the fitted curve is needed at (additional) positions x^*, then we can introduce pseudo-observations x^*, y^* with zero weights, and arbitrary values for y^*; smoothly interpolated (extrapolated) values are obtained automatically. In this paper we work with data on a grid (for x) and here the use of pseudo-observations preserves the grid structure of the data. This approach is easier than leaving out the missing data, fitting P-splines to the observed data and then computing the fitted curve at the missing positions.

Standard errors for the estimated coefficients, as well as for the fitted curve, can be computed as follows. The variance of e in the model $y = B\alpha + e$ is estimated as $s^2 = \hat{\mathrm{var}}(e) = \|y - B\hat{\alpha}\|^2/(m - ED)$, where m is the number of observations. The estimate of the covariance matrix of $\hat{\alpha}$ follows from (3) as

$$\hat{C}_1 = \hat{\mathrm{Var}}(\hat{\alpha}) = s^2 (B'B + \lambda D_d' D_d)^{-1} B'B (B'B + \lambda D_d' D_d)^{-1}, \tag{7}$$

or the simpler expression

$$\hat{C}_2 = \hat{\mathrm{Var}}(\hat{\alpha}) = s^2 (B'B + \lambda D_d' D_d)^{-1}. \tag{8}$$

Chapter 3 of Hastie and Tibshirani (1990), or Chapter 6 of Ruppert et al. (2003) have extensive discussions on how these two results can be obtained. We prefer $\hat{C}_2$ from Eq. (8) because it is the more conservative of the two; it is also easier to compute. For the covariance matrix of $\hat{\mu} = B\hat{\alpha}$ this gives $B\hat{C}_2 B' = s^2 H$, where H is the hat matrix. In practice, especially for large m, we will only be interested in the diagonal of H, from which point-wise confidence bands for the fitted curve $\hat{\mu}$ can be computed.

Eilers and Marx (1996) showed that the above development can be extended to the generalized linear model (GLM) (McCullagh and Nelder, 1989). We assume that $y = \mu + e$, $\mu = E(y)$, where the distribution of e is a member of the exponential family, such as the Poisson or binomial distribution. We suppose the existence of a function $g(\cdot)$, the link function, and a linear predictor η such that

$$g(\mu) = B\alpha = \eta. \tag{9}$$

For the Poisson distribution the logarithmic link, $g(\mu) = \log(\mu)$, is often appropriate; the logistic link, $g(\mu) = \log(p/(1 - p))$, plays the corresponding role for the binomial distribution. Both these link functions have attractive mathematical properties and are known as the natural link functions for their respective error distributions. In place of the penalized least squares criterion in (1) we use the penalized log-likelihood

$$l^*(\alpha) = l(\alpha) - \lambda \|D_d\alpha\|^2/2, \tag{10}$$

which leads to the penalized likelihood equations

$$B'(y - \mu) = \lambda D_d' D_d \alpha. \tag{11}$$

This system is non-linear in α, but by using the scoring method we arrive at iterations of the following linearized system

$$(B'\tilde{W}B + \lambda D_d' D_d)\alpha = B'(y - \tilde{\mu}) + B'\tilde{W}B\tilde{\alpha}. \tag{12}$$

Here a tilde as in $\tilde{\alpha}$ indicates an approximate solution. The matrix of weights *W* is diagonal, with elements $w_{ii}^{-1} = (\partial\eta_i/\partial\mu_i)^2 \mathrm{var}(y_i)$.

It remains to choose an optimal value of λ in this penalized GLM setting. The Akaike information criterion, $\mathrm{AIC} = -2l(\hat{\alpha}) + 2ED$, is convenient to compute since $l(\hat{\alpha})$ is given by (10) and *ED*, the effective dimension, by

$$\begin{aligned} ED = \mathrm{tr}(H) &= \mathrm{tr}[B(B'\hat{W}B + \lambda D_d' D_d)^{-1} B'\hat{W}] \\ &= \mathrm{tr}[(B'\hat{W}B + \lambda D_d' D_d)^{-1} B'\hat{W}B]. \end{aligned} \tag{13}$$

P-splines are an effective tool for one-dimensional smoothing of many types of data like scatterplots, densities (histograms), and dose-response curves. In the next section we show how to extend P-splines to two-dimensional data, using tensor products of B-splines.

3. Scattered data in two dimensions

Tensor products allow a natural extension of one-dimensional P-spline smoothing to two dimensions. We assume that in addition to *x* we have a second explanatory variable *v*. We have data triples (v_i, x_i, y_i) for $i = 1, ..., m$. We seek a smooth surface $\mu(v, x)$ which gives a good approximation to the data. Let *B*, $m \times L$, be a B-spline basis along *x*, and $\breve{B}$, $m \times K$, be a B-spline basis along *v*. We form the *KL* tensor products $T_{kl}(v, x) = \breve{B}_k(v)B_l(x)$, $k = 1, \ldots, K$ and $l = 1, \ldots, L$.

Let $A = [\alpha_{kl}]$ be a $K \times L$ matrix of coefficients. Then, for given *A*, the fitted value at (v, x) is

$$\mu(v, x) = \sum_k \sum_l \breve{B}_k(v) B_l(x) \alpha_{kl} \tag{14}$$

and so *A* may be chosen by least squares by minimizing

$$S = \sum_i [y_i - \mu(v_i, x_i)]^2 = \sum_i \left[y_i - \sum_k \sum_l \breve{B}_k(v_i) B_l(x_i) \alpha_{kl} \right]^2. \tag{15}$$

This is the two-dimensional equivalent of unpenalized B-spline regression; see Dierckx (1993) for an extensive discussion. To gain the advantages of P-splines, we introduce the following penalty function:

$$\mathrm{Pen}(A) = \lambda \sum_{l} \|D_d a_{\bullet l}\|^2 + \breve{\lambda} \sum_{k} \|a_{k\bullet} \breve{D}_{\breve{d}}\|^2. \tag{16}$$

The first term puts a difference penalty on each columns of A ($a_{\bullet l}$ stands for column l) and the second term puts a difference penalties on each row of A ($a_{k\bullet}$ stands for row k). We note two important features of (16): first, the order of the differences, d and $\breve{d}$ on rows and columns, respectively, can be different; second, the weights λ and $\breve{\lambda}$ can be different. This allows different amounts of smoothing along the two dimensions; an extreme example would have $\lambda = 0$ and $\breve{\lambda} = \infty$ which corresponds to polynomial regression in the x-direction (depending on the order of the differences on the rows) but very light smoothing along the y-direction.

It is possible to derive (penalized) normal equations, and then construct and solve them using a low-level language like C(++) or Fortran. Our aim here is more ambitious: we seek a fast implementation in a high level language like Matlab or S-PLUS/R, using only matrix and vector operations. To achieve this, we derive a regression basis C from B and $\breve{B}$, and flatten A, by stacking its columns, to a vector of regression coefficients β. We get C from the formula

$$C = \breve{B} \square B = (B \otimes e'_L) \odot (e'_K \otimes \breve{B}), \tag{17}$$

where the operation $\square$ is defined in terms of the Kronecker product of two matrices (denoted by $\otimes$) and the element-by-element multiplication of two matrices (denoted by $\odot$); e_K and e_L are vectors of ones of length K and L. Note that the operations in (17) are such that row i of C is the Kronecker product of the corresponding rows of $\breve{B}$ and B. The similar column-wise procedure is known as the Khatri–Rao product (Rao and Mitra, 1971).

With these definitions of C and β we are back at regression: $E(y) = \mu = C\beta$. We also construct penalty matrices P and $\breve{P}$ that express the two terms of the penalty on A in (16) in terms of penalties on β. We find

$$\mathrm{Pen}(A) = \lambda \beta' P \beta + \breve{\lambda} \beta' \breve{P} \beta \tag{18}$$

where it can be shown that $P = I_L \otimes D'_d D_d$ and $\breve{P} = \breve{D}'_{\breve{d}} \breve{D}_{\breve{d}} \otimes I_K$, where I_K and I_L are identity matrices of sizes K and L, respectively. So, with weights W, we solve

$$(C'WC + \lambda P + \breve{\lambda} \breve{P})\hat{\beta} = C'Wy. \tag{19}$$

Finally we can recover the grid form, $\hat{A}$, of the regression coefficients by rearranging $\hat{\beta}$ into L columns of length K.

This algorithm is suitable for small or moderate amounts of (scattered) data. There is no condition that x and y should be on a grid. The computations for effective dimension, deletion residuals and standard errors are exact parallels of those for the one-dimensional case, $C'WC$ taking the place of $B'WB$ and $\lambda P + \breve{\lambda}\breve{P}$ taking the place of $\lambda D'_d D_d$. It can also be extended to three or more dimensions, but we will not discuss the details here.

However, with large data sets we get into trouble. In Section 6 we present an example with data on a 500×500 grid. We use a basis which is the tensor product of 22 basis functions in the *x*-direction and 22 in the *v*-direction. The regression matrix has over 1.2×10^8 elements; if each floating point number takes up 8 bytes, *C* will use nearly 1 Gb memory. The computation of $C'WC$ makes even further demands. This example seems beyond the reach of current computers. In the next section we look at a novel algorithm that solves this storage problem for data on a grid. This discussion has concentrated on storage problems. We will see that our algorithm also gives savings in computing time of orders of magnitude over conventional regression methods.

4. Data on a two-dimensional grid

Let the data be in a matrix *Y*, with *I* rows and *J* columns. We suppose that the rows have coordinates v_i, $i = 1, \ldots, I$, and the columns coordinates x_j, $j = 1, \ldots, J$. In many cases these coordinates will be implicit, e.g., smoothing an image, when $v_i = i$ and $x_j = j$ are natural choices. Let *B*, $J \times L$, be a basis matrix with elements $b_{jl} = B_l(x_j)$ and let $\breve{B}$, $I \times K$, be a basis matrix with elements $\breve{b}_{ik} = \breve{B}_k(v_i)$. Let *A* be the $K \times L$ matrix of tensor product coefficients. Then, corresponding to (14), we have that

$$\mu_{ij} = \sum_k \sum_l \alpha_{kl} \breve{b}_{ik} b_{jl}. \tag{20}$$

We now take advantage of the fact that the data are arranged in a matrix *Y* to rewrite (20) as

$$M = E(Y) = \breve{B} A B'. \tag{21}$$

This simple formula provides the key to our method. If we express this problem as a conventional regression problem, as in the previous section, then the regression matrix $C = B \otimes \breve{B}$ (Searle, 1982, p. 333). Thus (21) allows us to compute $C\beta$ without the evaluation of *C*; this solves the storage problem. Further analysis shows that the number of multiplications in (21) is an order of magnitude smaller than required in the direct evaluation of $C\beta$; this gives our algorithm its speed. We now extend this idea to the efficient evaluation of the normal equations. In the next section we generalize our method to higher dimensions.

We consider fitting the model in the least squares sense, without a penalty, but with weights $W = [w_{ij}]$. We have to minimize

$$S = \sum_i \sum_j w_{ij} \left(y_{ij} - \sum_k \sum_l \alpha_{kl} \breve{b}_{ik} b_{jl} \right)^2. \tag{22}$$

The normal equations follow from $\partial S / \partial \alpha_{\bar{k}\bar{l}} = 0$. We find

$$\sum_k \sum_l \left(\sum_i \sum_j w_{ij} \breve{b}_{ik} b_{jl} \breve{b}_{i\bar{k}} b_{j\bar{l}} \right) \hat{\alpha}_{kl} = \sum_i \sum_j w_{ij} y_{ij} \breve{b}_{i\bar{k}} b_{j\bar{l}},$$
$$\bar{k} = 1, \ldots, K, \quad \bar{l} = 1, \ldots, L. \tag{23}$$

The right-hand side of this equation is easy to compute since it is equivalent to

$$R = \breve{B}'(W \odot Y)B, \tag{24}$$

as in (21), but the left-hand side has no direct simple expression. We rewrite (23) as

$$\sum_k \sum_l f_{k\bar{k}l\bar{l}} \hat{\alpha}_{kl} = r_{\bar{k}\bar{l}}, \quad \bar{k} = 1, \ldots, K, \ \bar{l} = 1, \ldots, L, \tag{25}$$

where $f_{k\bar{k}l\bar{l}} = \sum_i \sum_j w_{ij} \breve{b}_{ik} b_{jl} \breve{b}_{i\bar{k}} b_{j\bar{l}}$, for $k = 1, \ldots, K$, $\bar{k} = 1, \ldots, K$, $l = 1, \ldots, L$ and $\bar{l} = 1, \ldots, L$. Note that F is not a matrix, but a four-dimensional array with dimension $K \times K \times L \times L$. We reorder the sums for $f_{k\bar{k}l\bar{l}}$ as

$$f_{k\bar{k}l\bar{l}} = \sum_i \breve{b}_{ik} \breve{b}_{i\bar{k}} \sum_j w_{ij} b_{jl} b_{j\bar{l}}. \tag{26}$$

Let G be a $J \times L^2$ matrix with all products of the rows of the basis functions in B. This is our $\Box$ function defined in (17). We have

$$G = B \Box B = (e'_L \otimes B) \odot (B \otimes e'_L), \tag{27}$$

In a similar way we let $\breve{G}$ be an $I \times K^2$ matrix defined by

$$\breve{G} = \breve{B} \Box \breve{B} = (e'_K \otimes \breve{B}) \odot (\breve{B} \otimes e'_K). \tag{28}$$

Let $F^* = \breve{G}' W G$. Then F^* is a $K^2 \times L^2$ matrix and it is a simple matter to check that F^* contains exactly the same elements as the four-dimensional array F. In the process we constructed G and $\breve{G}$ which are of moderate size. If I and J are 1000, and B and $\breve{B}$ are both 1000×20, then C would be $10^6 \times 400$. With the present approach both G and $\breve{G}$ have 1000 rows and 400 columns and F^* is computed with only a little over 3 Mb of memory.

The final step is to express the normal equations (25) in conventional matrix form. We reorder the elements of F^* into a $KL \times KL$ matrix T such that if $\beta = \text{vec}(A)$ and $q = \text{vec}(R)$, the desired solution follows from $T\hat{\beta} = q$. This can be done efficiently by repeatedly permuting dimensions and re-dimensioning, as will be shown below. The computation of T and q is done independently of the penalty (18) which can be added to T as in (19).

We explain the re-ordering operations using Matlab notation. We assume the availability of a function `box()` that computes the row-wise Kronecker products: $box(B, B) = B \Box B$. The data are in matrices `Y` and `W`, with `m1` rows and `m2` columns; the basis matrices are `B1`, of size `m1` by `n1`, and `B2`, of size `m2` by `n2`; the penalty matrix `P`, of size `n1*n2` by `n1*n2`, has been computed already. The code in Algorithm 1 constructs the required inner products, solves the system, and delivers the coefficients in the `n1` by `n2` matrix `A`. Note that in Matlab `*` stands for matrix multiplication and `.*` for element-by-element multiplication, and that the apostrophe, as in `B'`, indicates matrix transposition.

Algorithm 1.

```
R = B1'*(W.*Y)* B2;
r = reshape(R,[n1*n2,1]);                % Make it a vector
F0 = box(B1,B1)'* W * box(B2,B2);
F1 = reshape(F0,[n1,n1,n2,n2]);          % Make 4-D array F1
F2 = permute(F1,[1,3,2,4]);              % Permute dimensions
F3 = reshape(F2,[n1*n2,n1*n2]);          % Back to 2-D
a = (F3+P)\ r;                           % Solve penalized system
A = reshape(a,[n1,n2]);                  % Make it a matrix
```

Different variables have been introduced here for clarity; in an actual implementation only one `A`, `F` and `R` is required since reshaping and permutating can be done in situ, and intermediate results are not needed any more. Users of Splus/R should modify the above program with `dim()` and `aperm()` replacing Matlab's `reshape()` and `permute()`.

Readers may find it helpful to compare the above calculation with the standard solution (Algorithm 2) to the normal equations.

Algorithm 2.

```
C = kron(B2,B1);                            % The regression basis
w = reshape(W,[m1*m2,1]);
y = reshape(Y,[m1*m2,1]);
CW = C.*repmat(w,1,n1*n2);                  % Weight rows of C by w
CWC = CW'*C;                                % Inner products
CWy = CW'*y;                                % Right hand side
a2 = (CWC+P)\ CWy;
disp(max(abs(a-a2)));                       % Compare answers
```

In the first program we avoid computation of the regression matrix C and compute the inner products $C'WC$ and the linear functions $C'Wy$ with matrix multiplications involving only `B1` and `B2`. This contrasts with the check program where we start by computing C.

The bottom line is that we can compute a weighted smooth of a 1000×1000 data matrix with a regression basis of 400 tensor products in 4 s on a 1 Ghz Pentium III PC. There is even room for further improvement: in $G = B \Box B$ and $\breve{G} = \breve{B} \Box \breve{B}$ each product of basis functions occurs twice, as in $b_{jl} b_{j\bar{l}}$ and $b_{j\bar{l}} b_{jl}$. By juggling the column indices when building G, we can nearly halve the number of columns in G from L^2 to $L(L+1)/2$. Since the same holds for $\breve{G}$, we can nearly quadruple the speed of the computations; of course, we need additional un-juggling steps when building T from F^*.

We postpone the computation of standard errors of fitted values to the next section, where we analyze the general case of three dimensions and higher.

5. Extension to higher dimensions

We extend the ideas of the preceding section to three dimensions and higher. We do not feel that it is helpful to attempt to write down the normal equations in higher dimensions corresponding to (23) in two dimensions. Instead we will present the Matlab implementation. In contrast to the algebraic approach, the computations in Matlab are straightforward, and patterns should be clear in any number of dimensions. Moreover, these computations are easily checked with programs similar to the one used in the previous section.

We need to generalize the product of two matrices to the product of a matrix and a multidimensional array. We illustrate with the case of a three-dimensional array. Let A be a matrix and B be an array with 3 dimensions. We define the function ρ in the following way:

$$C_1 = \rho(A, B, 1) \Rightarrow c_{hkl} = \sum_j a_{jh} b_{jkl}; \tag{29}$$

$$C_2 = \rho(A, B, 2) \Rightarrow c_{jhl} = \sum_k a_{kh} b_{jkl}; \tag{30}$$

$$C_3 = \rho(A, B, 3) \Rightarrow c_{jkh} = \sum_l a_{lh} b_{jkl}. \tag{31}$$

Notice first that $\rho(A, B, p)$ is another three-dimensional array with the same dimensions as B except that its pth dimension is that of the column-dimension of the matrix A. In words: $\rho(A, B, p)$ performs the usual computations of the matrix product, along the rows of A and along dimension p of B; of course, the first dimension of A and the pth dimension of B must be of the same size. A Matlab implementation of this function can be written in a few lines. The trick is to rotate dimension p to the front by dimension permutation, reduce the array to two dimensions, compute the standard matrix product with A', transform back to three dimensions and rotate the pth dimension back into place. A Matlab implementation of the function `rho()` is shown as Algorithm 3.

Algorithm 3.

```
function C = rho(A,B,p);
sa = size(A);
sb = size(B);
n = length(sb);
ip = [(p+1):n,1:(p-1)];
B = permute(B,[p,ip]);
B = reshape(B,[sb(p),prod(sb(ip))]);
C = A'* B;
C = reshape(C,[sa(2),sb(ip)]);
C = ipermute(C,[p,ip]);
```

Having defined `rho()`, we can now proceed in a similar way to two dimensions. We suppose that the data are in `Y` and the weights in `W`, both with dimensions `m1` by `m2` by `m3`. The basis matrices are `B1`, of size `m1` by `n1`, `B2`, of size `m2` by `n2`, and `B3`, of size `m3` by `n3`. As before, the function `box()` computes the row-wise tensor products and `P` contains the penalties. See Algorithm 4.

Algorithm 4.

```
F = rho(box(B1,B1),W,1);
F = rho(box(B2,B2),F,2);
F = rho(box(B3,B3),F,3);
R = rho(B1,Y.* W,1);
R = rho(B2,R,2);
R = rho(B3,R,3);
F = reshape(F,[n1,n1,n2,n2,n3,n3]);
F = permute(F,[1,3,5,2,4,6]);
F = reshape(F,[n1*n2*n3, n1*n2*n3]);
A = (F+P)\ reshape(R,[n1*n2*n3, 1]);
A = reshape(A,[n1,n2,n3]);
```

In 3 dimensions, the penalty `P` is computed as in Algorithm 5 (the function `kron()` computes Kronecker products while the function `eye(n)` delivers an identity matrix of size `n`):

Algorithm 5.

```
E1 = eye(n1);E2 = eye(n2);E3 = eye(n3);
D1 = diff(E1,d1);D2 = diff(E2,d2);D3 = diff(E3,d3);
P1 = kron(kron(D1'* D1,E2),E3);
P2 = kron(kron(E1,D2'* D2),E3);
P3 = kron(kron(E1,E2),D3'* D3);
P = lambda1 * P1+lambda2 * P2+lambda3 * P3;
```

It should be clear how to generalize these procedures to higher dimensions: we apply `rho()` to each dimension, extend the lists of coordinates in the reshaping and permutation operations, and nest more Kronecker products in the computation of the penalties. With some care it should be possible to make a general procedure which accepts as input the data and a cell array (a Matlab equivalent for a list) containing the bases.

It remains to provide efficient computation of the diagonal of the hat matrix H corresponding to a flattened tensor product basis. In three dimensions it can be implemented as in Algorithm 6 (we catch up with the last line that reshaped F, which we repeat here):

Algorithm 6.

```
F = reshape(F,[n1*n2*n3, n1*n2*n3]);
G = inv(F+P);
G = reshape(G,[n1,n2,n3, n1,n2,n3]);
G = permute(G,[1,4,2,5,3,6]);
G = reshape(G,[n1*n1, n2*n2, n3*n3]);
H = rho(B1,G,1);
H = rho(B2,H,2);
H = rho(B3,H,3);
```

An amazing property of this algorithm is that it only computes the diagonal of H. In full, H is an $m_1 m_2 m_3 \times m_1 m_2 m_3$ matrix, but we compute only the desired elements in an $m_1 \times m_2 \times m_3$ array. The proof requires careful manipulations of the sums that form the required elements; we omit it here. As usual, a check program is easily written.

In higher dimensions one has to reduce the number of basis functions drastically to get a manageable number of tensor products. This makes sense anyway, as huge amounts of data are needed to estimate interactions reliably in, say, four or five dimensions. A sensible rule of thumb could be to arrange that the total number of tensor product basis functions is around 1000; if all dimensions are of comparable size, this is achieved by using around $\sqrt[d]{1000}$ basis functions per dimension in a d-dimensional application.

As an interesting aside, we present in Algorithm 7 the computations for *one* dimension, showing the generality of the algorithms. Now the data are vectors `y` and `w` of length `m` and the basis `B` is `m` by `n`.

Algorithm 7.

```
F = rho(box(B,B),w,1);
r = rho(B,w.*y,1);
F = reshape(F,[n,n]);
a = (F+P)\ r;
G = inv(F+P);
g = reshape(G,[n*n, 1]);
h = box(B)*g;                                   % Diagonal of H
```

6. Applications

We use our method on three examples. The first example is a simulated noisy image of 500×500 pixels. In the second example we smooth a mortality table with data from the insurance industry. The final example returns to the first example with the twist that, because of sampling, the data are no longer on a grid; we show how our methods can still be applied in this case.

The left panel of Fig. 1 gives a plot of the image data. The height of the simulated surface is indicated by the intensity of the image with a light colour indicating a greater height. Noise was added to the image and all weights were equal to 1. The image of lunar craters formed by meteor bombardments may be helpful. The enhanced image in the right panel was produced by smoothing with a grid of 22×22 tensor products of quadratic B-splines and third order differences. The smoothing parameters were $\lambda = \breve{\lambda} = 1$.

Fig. 3 shows an application involving a “mortality table” for life insurance policies in the UK, covering the period from 1947 to 1999 and ages from 21 to 100. If y_{ij} is the number of policies terminating in year i for persons of age j, and e_{ij} is the exposure, i.e., the number of years at risk for persons of age j in year i, then smooth (extrapolated) estimates of the raw mortality ratios y_{ij}/e_{ij} are of prime importance to insurance companies. See Durban et al. (2002) and Currie et al. (2003) for further details. We model these data as Poisson distributed with $E(y_{ij}) = \mu_{ij}$ and $\log \mu_{ij} = \log e_{ij} + \phi_{ij}$, with a sum of tensor products for

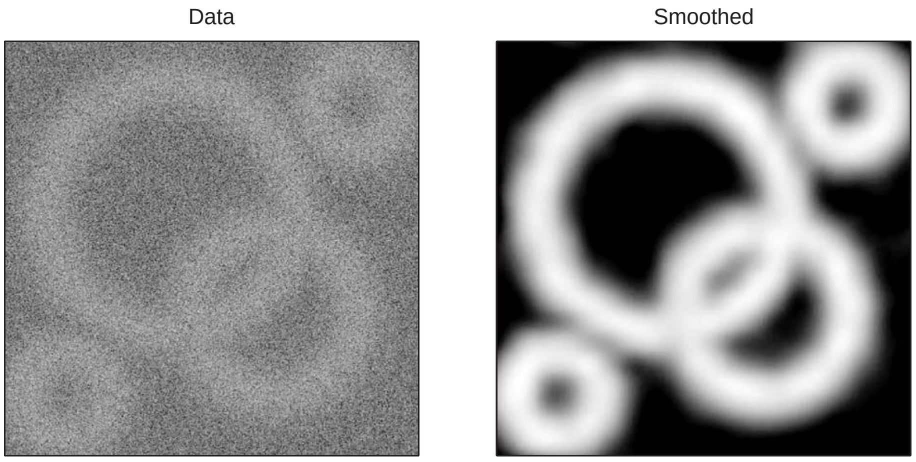

Fig. 1. Left panel: a noisy image; right panel: the enhanced image.

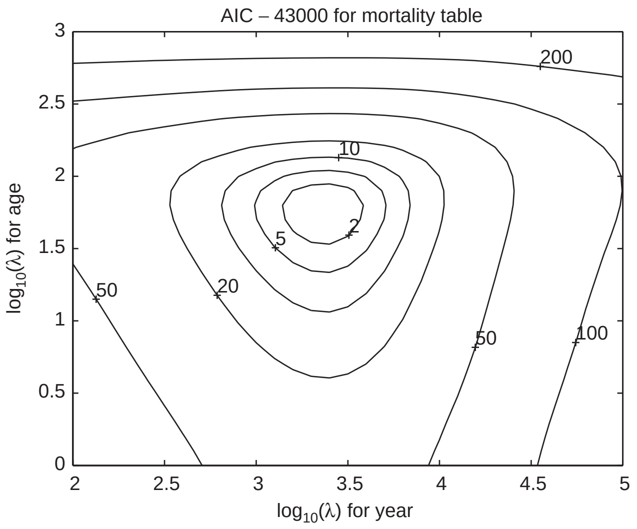

Fig. 2. Contour lines of AIC (minus 43000) for the mortality data. The search grid had a spacing of 0.1 (on the logarithmic scale).

the matrix Φ: $\Phi = \breve{B}AB'$. A basis of 12×12 tensor products of quadratic B-splines was used with a quadratic penalty. It takes less than a second to fit the model (for one set of values of the λs) on a 1 GHz Pentium III computer. We compute values of the AIC criterion over a grid of values of λ and $\breve{\lambda}$; a contour plot is shown in Fig. 2. The optimum is reached near $\lambda = 65$ (for age) and $\breve{\lambda} = 2500$ (for year). After smoothing, one can clearly see the steeply rising contour lines at higher ages, indicating rapidly improving mortality. Since

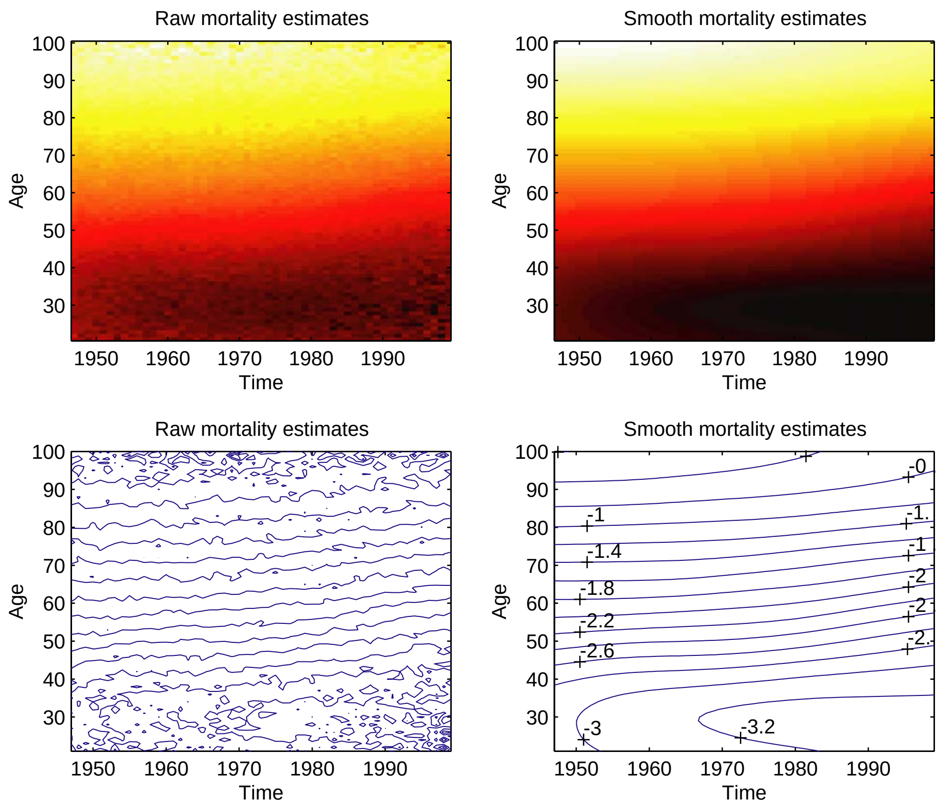

Fig. 3. Smoothing of a mortality table of life insurance policies in the UK. Top panels: plots with intensity proportional to the logarithm of mortality, raw (left) and smoothed (right). Bottom panels: contour lines for mortality on a logarithmic scale (base 10), raw (left) and smoothed (right).

pricing and reserving for many insurance products depends on accurate predictions of future mortality, these increases are a serious cause for concern for insurance companies.

The algorithm for data on a grid derives its speed from the regular pattern in the observations and it seems that we cannot have this benefit for scattered data. However, if we allow some slight inaccuracies, there is an elegant way out. We divide the x-v plane into rectangular bins and build two matrices: (1) W, which holds the number of observations in each bin; (2) S, which holds, for each bin, the average of the y values of the observations falling in that bin (or zero if there are none). We then smooth S with weights W. If small bins are used, say a 200×200 grid, the error induced by the discretization will be negligible. Fig. 4 shows an application. The (noise-free) image with the rings was sampled at 1000 random positions and these observations were discretized and smoothed. The data are shown in the left panel. A black pixel indicates no observation, while a grey pixel indicates one or more observations with intensity proportional to their average. The right panel shows the result of smoothing with a grid of 22×22 tensor products of quadratic B-splines with third order differences; again we set $\lambda = \breve{\lambda} = 1$.

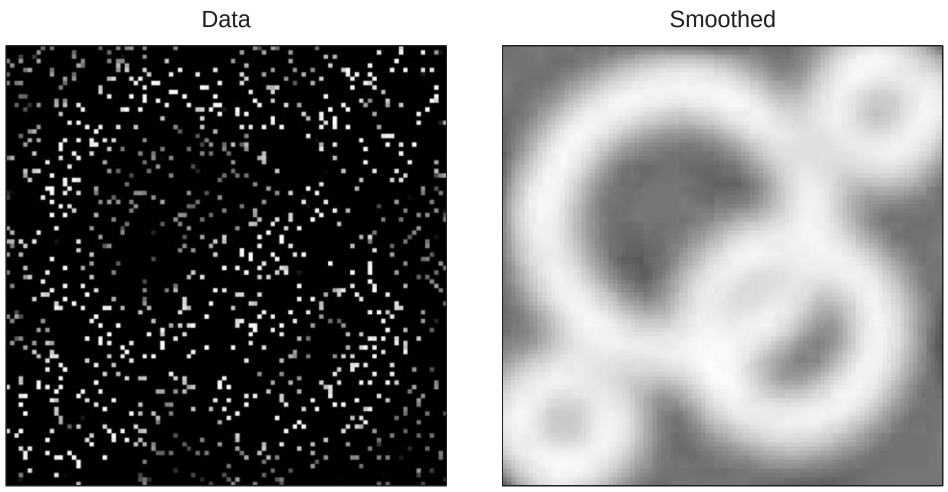

Fig. 4. Smoothing of scattered data using bins and the fast grid algorithm.

7. Discussion

We have presented a fast, low-footprint algorithm for smoothing of multidimensional data on a grid with arbitrary weights. We have sketched how to extend the algorithm to more than two dimensions and we have presented some two-dimensional applications. The algorithm has also been applied successfully to smoothing three-dimensional counts of mortality cross-tabulated by age, year and month. The periodic character of the month dimension introduces an interesting complication. Without special precautions, there may not be a smooth connection between December and the following January. We are experimenting with a “circular” B-spline basis and corresponding penalty which we construct by wrapping the basis functions around.

Extensions to sums of tensor product bases are possible. One extension uses two sets of tensor product bases to model agricultural experiments. One basis is built from B-splines and models yield, while the other basis contains 0–1 indicator functions and models the rectangular plots in the field on which different varieties of a crop are grown. A similar application is encountered when we correct for spatial trends in microarray fluorescence, where the effect of printing pins causes shifts of sub-grids.

The regression structure of the multidimensional smoother makes it directly suitable to generalize the GLASS framework of Eilers and Marx (2002). Generalized additive models, varying coefficient models and multidimensional signal regression (Eilers and Marx, 2003) can be combined in a mix-and-match way to build complicated models for normal and non-normal responses.

Unfortunately, the algorithm is not suitable for light smoothing of large matrices, since this would imply too many basis functions for the tensor products. The system of penalized normal equations then becomes the bottleneck. A possible, but inelegant, solution would be to smooth and combine (overlapping) sub-arrays. Further research is needed in this area.

We are researching the application of mixed model algorithms as an alternative to AIC for smoothing parameter selection. Ruppert et al. (2003) give a detailed account of this approach in one dimension using truncated power function (TPF) bases; they report some difficulties in extending this to higher dimensions. The mixed model approach is also available with B-spline bases (Eilers, 1999; Currie and Durban, 2002). In general, extensions to higher dimension with penalized tensor product bases seem likely to run into the same difficulties encountered with TPF bases. However, grid data and models with tensor product bases may provide sufficient extra structure to allow a mixed model representation. We have some positive results in particular cases and work continues on a general solution.

Matlab and S-PLUS/R code can be obtained from the first author.

Acknowledgements

The research of Maria Durban was supported by DGES project BEC 2001-1270.

References

Currie, I.D., Durban, M., 2002. Flexible smoothing with P-splines: a unified approach. Statist. Modelling 2, 333–349.

Currie, I.D., Durban, M., Eilers, P.H.C., 2003. Using P-splines to extrapolate two-dimensional Poisson data. Proceedings of 18th International Workshop on Statistical Modelling. Leuven, Belgium, pp. 97–102.

Dierckx, P., 1993. Curve and Surface Fitting with Splines, Clarendon Press, Oxford.

Durban, M., Currie, I.D., Eilers, P.H.C., 2002. Using P-splines to smooth two-dimensional Poisson data. Proceedings of 17th International Workshop on Statistical Modelling, Chania, Crete, pp. 207–214.

Eilers, P.H.C., 1999. Discussion of The Analysis of Designed Experiments and Longitudinal Data by Using Smoothing Splines, by Verbyla, A.P., Cullis, B.R., Kenward, M.G. and Welham, S.J.. Appl. Statist. 48, 269–311.

Eilers, P.H.C., Marx, B.D., 1996. Flexible smoothing with B-splines and penalties (with Discussion). Statist. Sci. 11, 89–121.

Eilers, P.H.C., Marx, B.D., 2002. Generalized linear additive smooth structures. J. Comput. Graphical Statist. 11, 758–783.

Eilers, P.H.C., Marx, B.D., 2003. Multivariate calibration with temperature interaction using two-dimensional penalized signal regression. Chemometr. Intell. Lab. Systems 66, 159–174.

Hastie, T., Tibshirani, R., 1990. Generalized Additive Models, Chapman & Hall, London.

McCullagh, P., Nelder, J.A., 1989. Generalized Linear Models, 2nd ed. Chapman & Hall, London.

Rao, C.R., Mitra, S.K., 1971. Generalized Inverse of Matrices and its Applications, Wiley, New York.

Ruppert, D., Wand, M.P., Carroll, R.J., 2003. Semiparametric Regression, Cambridge University Press, Cambridge.

Searle, S.R., 1982. Matrix Algebra Useful for Statistics, Wiley, New York.