Stable Evaluation of Box Splines

Leif Kobbelt

Computer Sciences Department, University of Wisconsin — Madison
1210 West Dayton Street, Madison, WI 53706-1685, USA

November 8, 1996

Abstract

The most elegant way to evaluate box-splines is by using their recursive definition. However, a straightforward implementation reveals numerical difficulties. A careful analysis of the algorithm allows a reformulation which overcomes these problems without losing efficiency. A concise vectorized MATLAB-implementation is given.

Introduction

In [2] the numerical problems which occur when evaluating box-splines by using the recurrence relation, are discussed. The solution proposed there is based on a test of reliability for intermediate results: If the point of evaluation x lies too close to one of the break-hyperplanes, the function value is tagged to be not valid and the function is evaluated at some "safe" point $x + \varepsilon$ instead. While this method allows to detect and prevent the *influence* of numerical difficulties, we propose a reformulation of the recursive algorithm which avoids the *occurrence* of such numerical problems. Further, it turns out that, in case of repeated directions, the computational complexity of the algorithm can be reduced significantly.

Box-Splines

We follow the notation used in [1]. Let the box-spline $M_\Xi : \mathbb{R}^s \to \mathbb{R}$ be given by the matrix $\Xi \in \mathbb{R}^{s \times n}$ with $n \geq s$. To evaluate M_Ξ at some point $x \in \mathbb{R}^s$, we apply the recurrence relation:

$$(n-s)\; M_\Xi(x) \;=\; \sum_{\xi \,\in\, \Xi} t_\xi \; M_{\Xi \setminus \{\xi\}}(x) \;+\; (1 - t_\xi)\; M_{\Xi \setminus \{\xi\}}(x - \xi) \tag{1}$$

where $x = \sum t_\xi \, \xi$ is a representation of x by some linear combination of the columns ξ of Ξ, e.g., the least norm solution $t = \Xi^T (\Xi \, \Xi^T)^{-1} \, x$. The base case of this recursion occurs when the matrix Ξ is square. In this case M_Ξ is the normalized characteristic function of the projected half-open unit box:

$$M_\Xi(x) = \frac{1}{|\det \Xi|} \, \chi_{\Xi \, \square}(x). \tag{2}$$

If $\operatorname{rank} \Xi < s$, we set $M_\Xi = 0$. The computational complexity of the evaluation can be measured by the number $C(n, s)$ of recursive procedure calls:

$$C(n,s) \;=\; 2n \cdot 2(n-1) \cdots 2\,(s+1) \;=\; 2^{n-s} \, \frac{n!}{s!}\,. \tag{3}$$

In the description of the algorithm we will use a slightly different definition: The spline will be represented by the matrix $\widehat{\Xi} \in \mathbb{R}^{s \times k}$ of pairwise distinct directions and their multiplicities will be given by a vector $\nu = (\nu_1, \ldots, \nu_k)$ with $\sum \nu_i = n$.

Computational Complexity

For the least norm representation $t = \Xi^T(\Xi\,\Xi^T)^{-1}\,x$, the coefficients t_ξ and t_η corresponding to identical directions $\xi = \Xi(:,i) = \eta = \Xi(:,j)$ agree. Consequently, some terms in the sum (1) occur multiple times. Combining these terms is equivalent to setting $t = \widetilde{\Xi}^T(\widetilde{\Xi}\,\widetilde{\Xi}^T)^{-1}\,x$, where $\widetilde{\Xi} := \widehat{\Xi}|_\nu$ is the matrix $\widehat{\Xi}$ with the columns corresponding to zero components of ν deleted. In terms of the box-spline definition via $\widehat{\Xi}$ and ν, we have

$$(\|\nu\|_1 - s)\; M_\nu\,(x) \;=\; \sum_{\nu_\xi \neq 0} t_\xi\; M_{\nu-\delta_\xi}\,(x) \;+\; (\nu_\xi - t_\xi)\; M_{\nu-\delta_\xi}\,(x-\xi)$$

with δ_ξ being the row of the $(k \times k)$ identity matrix corresponding to the position of ξ in $\widehat{\Xi}$. The complexity of this recursion in the worst case is

$$C(n,s) \;\leq\; 2k \cdots 2k \cdot 2(k-1) \cdots 2(s+1) \;\leq\; (2k)^{n-s},$$

which, especially for box-splines with $n > k$, is better than (3).

Numerical Instability

The numerical instability of the recursive algorithm stems from the fact that in the base case (2) the inside/outside-decision is based on intermediate results which are affected by rounding errors. For continuous functions the effect of such errors is kept in check by the modulus of continuity but the characteristic function $\chi_{\widetilde{\Xi}\,\square}$ is a step function.

The decision procedure is based on two inputs, the translated position vector $\tilde{x} := x - \xi_{i_1} - \cdots - \xi_{i_{n-s}}$ and a vector n_H orthogonal to the hyperplane $H = \mathrm{span}\{\xi_{j_1}, \ldots, \xi_{j_{s-1}}\} \subset \mathbb{R}^s$. Note that the problems arise not because of the rounding errors themselves but because of the fact that they may be different for semantically identical values $\tilde{x}_1$ and $\tilde{x}_2$: The value of $\tilde{x}$ is not independent of a permutation of the translations ξ_i since rounding errors accumulate during the iterative subtractions.

The same holds for the computation of n_H which therefore has to be independent of the ordering of the columns ξ_j and must not use other information than H. If this independence is not achieved then the rounding errors lead to random results for points $\tilde{x}$ lying very close to the hyperplane H. However, it is important to notice that rounding errors do not have to be avoided. It is sufficient to make them *consistent* as has already been noted in [2].

The easiest way to compute $\tilde{x}$ consistently is to delay all translations until the base case is reached. We introduce a new vector $\mu = (\mu_1, \ldots, \mu_k)$ which is initialized to be zero and rewrite the recursion

$$(\|\nu\|_1 - s)\; M_{\nu,\mu}\,(x) \;=\; \sum_{\nu_\xi \neq 0} t_\xi\; M_{\nu-\delta_\xi,\mu}\,(x) \;+\; (\nu_\xi - t_\xi)\; M_{\nu-\delta_\xi,\mu+\delta_\xi}\,(x) \tag{4}$$

in terms of $M_{\nu,\mu}\,(x) := M_\nu\,(x - \widehat{\Xi}\,\mu)$. The base case takes the form

$$M_{\nu,\mu}(x) = \frac{1}{|\det \widetilde{\Xi}|}\,\chi_{\widetilde{\Xi}\,\square}(x - \widehat{\Xi}\,\mu).$$

The supports of the functions M_{ν_1} and M_{ν_2} with corresponding direction matrices $\widetilde{\Xi}_1 := \widehat{\Xi}|_{\nu_1} = [\xi_{j_0}, \ldots, \xi_{j_{s-1}}]$ and $\widetilde{\Xi}_2 := \widehat{\Xi}|_{\nu_2} = [\xi_{j_1}, \ldots, \xi_{j_s}]$ share the hyperplane $H = \mathrm{span}\{\xi_{j_1}, \ldots, \xi_{j_{s-1}}\}$. In order to guarantee that the decision on which side of H a given point $\tilde{x}$ lies is made consistently, we have to make sure that the same vector $n_H \in H^\perp$ is used everywhere. Since the specific path in the graph of the modified recursion (4) leading to a node ν does not affect the matrix $\widehat{\Xi}|_\nu$, we can use any deterministic procedure that computes n_H from $\widetilde{\Xi} := \widehat{\Xi}|_{\nu_1-\delta_{j_0}} = \widehat{\Xi}|_{\nu_2-\delta_{j_s}}$ alone. In our MATLAB-implementation, we use the built-in function `null(`$\widetilde{\Xi}^T$`)` which computes a basis for the kernel of $\widetilde{\Xi}^T$.

Optimization

The efficiency of the algorithm greatly improves if the normal vectors n_H are computed in advance and stored in a hash table with 2^k entries. In the base case (when $\widehat{\Xi}|_\nu$ is square), $\nu \in \{0,1\}^k$, because $\nu_i > 1$ would imply $\operatorname{rank}(\widehat{\Xi}|_\nu) < s$ and $M_\nu = 0$. Therefore we can interpret the vector ν as a binary number and use its value as the hashing function.

Other optimizations are possible if we do not consider the most general case of box-splines. A common restriction is that the matrix $\widehat{\Xi}$ have integer entries. In this case, the translations $x-\xi_i$ can be computed without rounding errors and therefore the delayed translation is not necessary.

Another restriction, usually made for $s = 2$, is that every subset of s columns of $\widehat{\Xi}$ be a basis for $\mathbb{R}^s$. In this case, the computationally expensive evaluation of $\operatorname{rank}(\widehat{\Xi}|_\nu)$ can be replaced by counting the non-zero entries of ν.

MATLAB-Implementation

A concise MATLAB-implementation of the proposed algorithm will be available in the NUMERALGO library. It consists of three function modules: `box_eval()`, `box_normals()` and `box_rec()`. The user interface is provided by the function `box_eval()` while the other two functions perform the recursive computation of the table of normal vectors and the recursive evaluation of the spline respectively.

The function `box_eval(X,nu,p)` encapsulates the calls to the other functions and computes some global data. The user passes a $(s \times k)$-matrix `X` $(= \Xi)$ whose column vectors are the distinct directions in $\mathbb{R}^s$ defining the box-spline (more precisely: the grid where the box-spline is living on). The second argument, `nu`, is an array of integers reporting the multiplicities of the directions of `X`. The last argument `p` is an array of points in $\mathbb{R}^s$ where the spline is to be evaluated. The procedure returns a vector of function values according to the ordering in `p`. The computational costs per box-spline evaluation decrease significantly with increasing number of points in `p`.

The function `box_normals(t,k,M)` recursively constructs the hash table of normal vectors. We want to interpret the index vector $\nu \in \{0,1\}^k$ as a k-bit binary number which identifies the normal vector corresponding to the hyperplane spanned by the column vectors ξ_i of Ξ (= row vectors of `BoxEv_X`) with $\nu_i = 1$. The semantic of the three arguments is as follows: `t` is the number of vectors that yet have to be selected in order to obtain a set of $s-1$ vectors spanning a hyperplane H in $\mathbb{R}^s$. During the recursion every candidate (= row of `BoxEv_X`) is considered and k indicates which one is next. Finally the bit-vector `M` indicates which rows already have been selected. At the root of the recursion tree, we have `t` $= s-1$, `k` $= k$ and `M` $= [0 \ldots 0]$.

If at some node in the recursion tree `k` $<$ `t` then no valid leaf occurs on this subtree and the recursion is aborted. Otherwise the table is split into the left and right half such that the addresses of the normal vectors reflect the value of the `k`th bit of a binary number.

The function `box_rec(n,m,Y,t)` is the implementation of the modified recursive evaluation algorithm. To minimize the number of transpose operations, this implementation uses the matrix `BoxEv_X` $= \Xi^T$ with the directions as *rows* vectors. The argument `m` $(= \mu)$ is passed through the recursion to register the current position within the recursion tree. This information is necessary to perform the delayed translation. For optimization purposes, the least norm representation `t` $(= t)$ is passed as an argument instead of recomputed. The same holds for the matrix `Y` which solves the normal equation for t. Both, `t` and `Y`, have to be updated only when a direction vector drops out of the matrix `BoxEv_X`. This happens when one of the entries `nu[i]` $(= \nu_i)$ turns zero during the recursion.

To demonstrate the use of the function `box_eval()`, an additional script-file, `demo.m`, is included. Three examples for box-splines on $\mathbb{R}^2$ are shown: the biquadratic tensor product Box-spline (Fig. 1), the Zwart-Powell-element (Fig. 2) and the second order Courant-element (Fig. 3).

Acknowledgements

I would like to thank Carl de Boor for acquainting me with the problem and for helpful discussions.

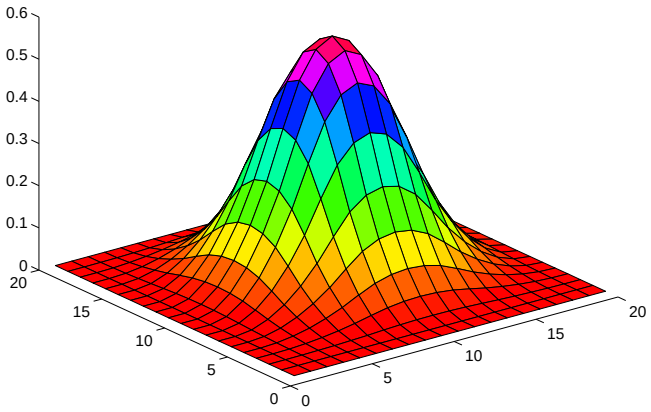

Figure 1: Biquadratic tensor product Box-spline: $\Xi = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, $\nu = (3\ 3)$.

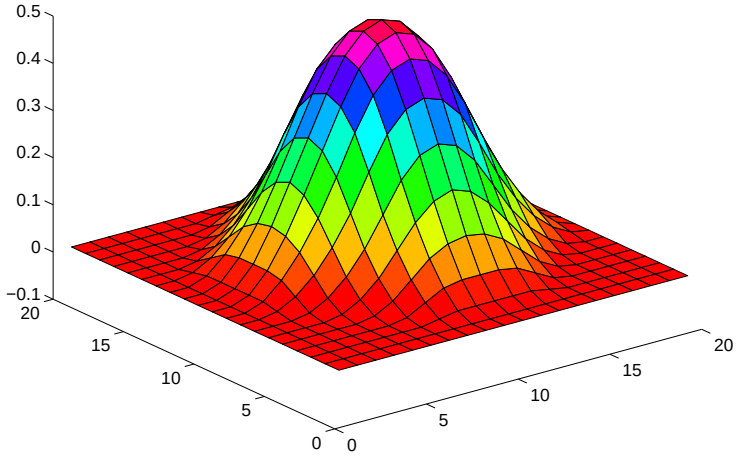

Figure 2: Zwart-Powell element: $\Xi = \left(\begin{smallmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & -1 & 1 \end{smallmatrix}\right)$, $\nu = (1\ 1\ 1\ 1)$.

References

[1] C. de Boor / K. Höllig / D. Riemenschneider, Box Splines, Springer Verlag Berlin (1993)

[2] C. de Boor, On the evaluation of box splines, Numer. Algorithms 8 (1993) 5–23

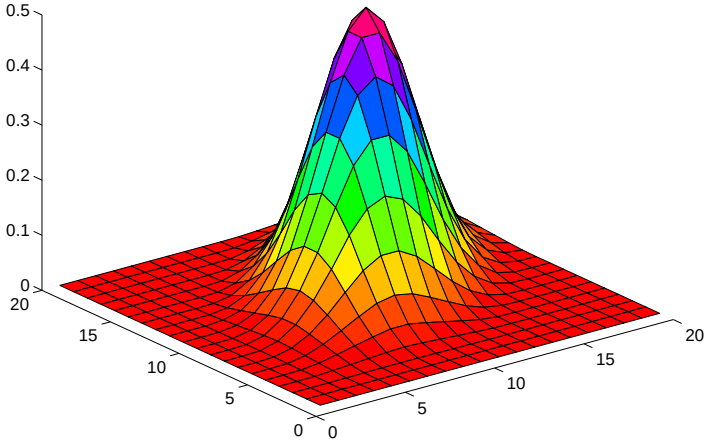

Figure 3: Second order Courant element: $\Xi = \left(\begin{smallmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{smallmatrix}\right)$, $\nu = (2\ 2\ 2)$.