Programming Multigrid Methods with B-Splines

Klaus Höllig, Jörg Hörner

Preprint 2011/016

Universität Stuttgart

Fachbereich Mathematik

Programming Multigrid Methods with B-Splines

Klaus Höllig, Jörg Hörner

Preprint 2011/016

Fachbereich Mathematik
Fakultät Mathematik und Physik
Universität Stuttgart
Pfaffenwaldring 57
D-70 569 Stuttgart

E-Mail: `preprints@mathematik.uni-stuttgart.de`
WWW: `http://www.mathematik.uni-stuttgart.de/preprints`

ISSN **1613-8309**

Programming Multigrid Methods with B-Splines

Klaus Höllig and Jörg Hörner

August 24, 2011

Abstract

Multigrid algorithms are the method of choice for solving large Ritz-Galerkin systems for elliptic boundary value problems. Using b-spline bases provides geometric flexibility as well as many computational advantages and permits particularly efficient and elegant implementations. This is described for a fairly general discretization which covers all principal features of b-spline elements.

1 Introduction

Multigrid algorithms generally provide the most efficient solvers for elliptic Ritz-Galerkin systems. The classical article by Brandt [8] beautifully describes and in detail analyzes programming strategies for numerous applications. The mathematical theory is fascinating, too. We refer, e.g., to the early convergence proofs of Fedorenko [14] and Bakhvalov [2] (difference equations) as well as Bank and Dupont [3] (finite elements). By now, there exists a vast literature on almost any aspect of multigrid algorithms (cf., e.g., [16, 9] and the references cited therein); multigrid solvers have become standard tools.

Due to their regular data structure, uniform b-splines are ideally suited for multilevel strategies. For weighted extended b-spline bases (web-splines), this has been demonstrated in [21]. In particular, a proof of the grid-independent convergence rate was given. For isogeometric elements, this should also be possible under appropriate assumptions. In this article, we consider a combination of these two principal b-spline-based approaches: weighted isogeometric elements, a mixed basis type introduced in [18]. This very general class of basis functions comprises the principal features of b-spline elements and has analogous properties. Hence, it is conceivable that the earlier multigrid convergence proofs can be adapted to the new setting. However, this is not the purpose of our article. Instead, we focus on the implementation of a typical multigrid scheme, showing that the simplicity and elegance of b-spline algorithms prevail also in a fairly complex setting.

Usually, finite element discretizations involve index lists and yield a linear system with irregular sparse structure which is solved with standard software. Proceeding in this way, many advantages of the b-spline basis are lost. Exploiting the regular

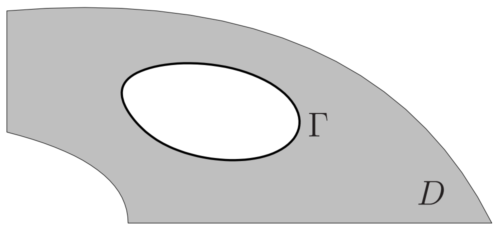

Figure 1: Domain $D \subset \mathbb{R}^2$ with inner boundary Γ

data structure not only leads to very efficient programs, these programs are also rather pretty from an esthetic point of view. If, in the end, the reader thinks that multigrid algorithms for b-splines are almost trivial to implement, the goal of our rather elementary exposition has been achieved: We would like to stimulate the interest in b-spline specific finite element programming.

A multigrid finite element algorithm with b-splines has, to our knowledge, first been programmed by Wipper [31]. Subsequently, multigrid methods with b-splines were used for several different applications [28, 19, 23, 24]. Our article builds on the experience gained by these earlier implementations and provides a general framework which applies to weighted as well as isogeometric elements. To keep the presentation short and easy to understand, we will neither use the most general rational b-spline basis nor consider a general elliptic boundary value problem. Moreover, we limit our discussion to discretizations in two variables. The generalizations are straightforward.

Our article is organized as follows. We begin in Section 2 by describing the Ritz-Galerkin discretization of a Poisson type equation with mixed boundary conditions which will serve as an elementary model problem. In Section 3 we introduce (polynomial) weighted isogeometric elements which combine essential features of the two main b-spline-based discretizations. Section 4 is devoted to a prototype of a dynamic multigrid algorithm. After these necessary prerequisites, we in detail discuss the implementation of the key multigrid modules in Sections 5 to 8 : matrix assembly, smoothing iteration, grid transfer, and the coarsening of Ritz-Galerkin matrices. Finally, we briefly comment on possible convergence proofs in Section 9.

2 Model Problem

For multigrid techniques, the Laplace operator often serves as a principal test case. Accordingly, we consider the mixed boundary value problem

$$-\Delta u + u = f \text{ in } D, \quad u = 0 \text{ on } \Gamma \subseteq \partial D, \quad \partial_\perp u = 0 \text{ on } \partial D \backslash \Gamma\,,$$

on a domain $D \subset \mathbb{R}^2$ of the form as in Figure 1 and with $\partial_\perp$ denoting the normal derivative. The particular form of the differential operator was chosen to allow $\Gamma = \emptyset$ as a special case. In contrast to Poisson's equation, there exists a unique solution also for pure natural boundary conditions ($\partial_\perp u = 0$ on ∂D).

The Ritz-Galerkin method (cf., e.g., [30, 32]) approximates the solution u by a linear combination

$$\sum_k u_k B_k$$

of suitable basis functions B_k (finite elements) which satisfy the essential boundary condition, i.e., $B_k = 0$ on Γ. The coefficients $U = (u_k)$ are computed with the aid of the weak form of the mixed boundary value problem:

$$\int_D \operatorname{grad} u \, \operatorname{grad} v \, + \, uv = \int_D fv, \quad \forall v \in H^1_\Gamma \, ,$$

where H^1_Γ is the Sobolev space of functions with square integrable first derivatives which vanish on Γ. Substituting $u = \sum_k u_k B_k$, $v = B_j$, leads to the linear system

$$\sum_k \left(\int_D \operatorname{grad} \, B_j \operatorname{grad} \, B_k \, + \, B_j B_k \right) u_k = \int_D B_j f \, ,$$

which we abbreviate as

$$GU = F \, .$$

While the Ritz-Galerkin approach is very general, it is clear that the choice of the basis functions is crucial. It influences the accuracy of the approximation as well as the efficiency of its computation.

3 Weighted Isogeometric B-Splines

B-splines have played a key role in many areas of applied mathematics and engineering (cf., e.g., [6, 29, 13, 11]), most recently also for finite element analysis. In this context, two different strategies have been proposed. Weighted approximations use uniform b-splines on a grid which covers the simulation region and incorporate essential boundary conditions via a weight function [20]. Isogeometric methods transform b-splines on tensor product grids to the physical domain via suitable parametrizations [22] (cf. the textbooks [17, 12] for a comprehensive description of the two approaches as well as [4, 1, 15] for samples of applications). Both methods have their pros and cons. Figure 1 gives an elementary example where each of the two techniques is not completely satisfactory. The weighted method does not exploit the geometric simplicity of the domain, whereas the isogeometric approach requires a partitioning which leads to unnecessary element distortions. A combination of both methods suggests itself and overcomes these difficulties. The basic principle is illustrated in Figure 2 for the particular example considered.

We parametrize the extended domain enclosed by the outer boundary by a smooth bijective mapping Φ which is defined on the unit square:

$$[0,1]^2 = Q \ni \xi \mapsto x = \Phi(\xi) \in D^{\mathrm{e}} \, .$$

Moreover, as in [20], we describe the inner boundary by a weight function $w\colon \, D^{\mathrm{e}} \to \mathbb{R}$ with

$$w = 0 \wedge \operatorname{grad} w \neq 0 \text{ on } \Gamma, \quad w > 0 \text{ in } \overline{D} \backslash \Gamma, \quad w < 0 \text{ in } \overline{D^{\mathrm{e}}} \backslash \overline{D} \, ,$$

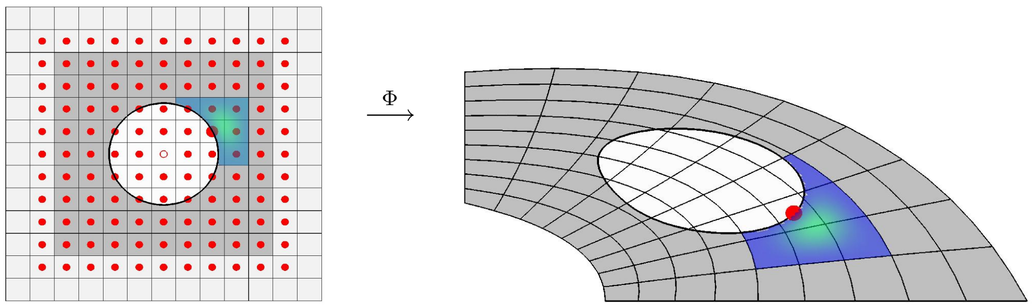

Figure 2: Weighted isogeometric b-splines

i.e., $D = \{x \in \Phi(Q) : w(x) > 0\}$. The discretization uses uniform b-splines b_k of coordinate degree n and grid width h, defined on Q. Denoting by M the numbers of grid cells in the two coordinate directions, $(M+n)^2$ b-splines are relevant, i.e., have some support in Q, namely:

$$b_k = b_{(k_1,k_2)}, \quad k_\nu = 1,\dots,M+n \quad (\nu = 1,2)\,,$$

where the subscript refers to the grid position. More precisely,

$$Q = [0, Mh] \times [0, Mh], \quad \operatorname{supp} b_k = [(k_1 - n - 1)h, k_1 h] \times [(k_2 - n - 1)h, k_2 h]\,.$$

With these definitions, the finite element basis for the mixed boundary value problem consists of so-called weighted isogeometric b-splines

$$x \mapsto B_k(x) = w(x) b_k(\xi), \quad \xi = \Phi^{-1}(x), \quad k \sim D\,,$$

which have some support in D (cf. [18] for a more detailed description). We note in particular that the multiplication by w ensures that B_k vanishes on Γ. Moreover, we remark that the pure weighted method corresponds to the special case $\Gamma = \partial D$, $\Phi = \mathrm{id}_{\mathbb{R}^2}$ for the sample problem under consideration. The pure isogeometric method can be recovered by choosing $\Gamma = \emptyset$, $w(x) = 1$.

For programming purposes, it is convenient to keep also irrelevant b-splines, i.e., elements B_k, $k \not\sim D$, which have no support in D. In Figure 2, relevant and irrelevant basis functions are distinguished by dots and circles in the center of their supports. This means, a weighted isogeometric spline has the form

$$\sum_{k_1=1}^{M+n} \sum_{k_2=1}^{M+n} u_k B_k\,.$$

Hence, an approximation is represented by a rectangular coefficient array

$$U : u_{(k_1,k_2)}, \quad 1 \le k_\nu \le M+n\,.$$

The coefficients u_k with $k \not\sim D$ can have arbitrary values since $B_k(x) = 0$ for $x \in D$. We will usually set them to zero or mark them with an appropriate flag.

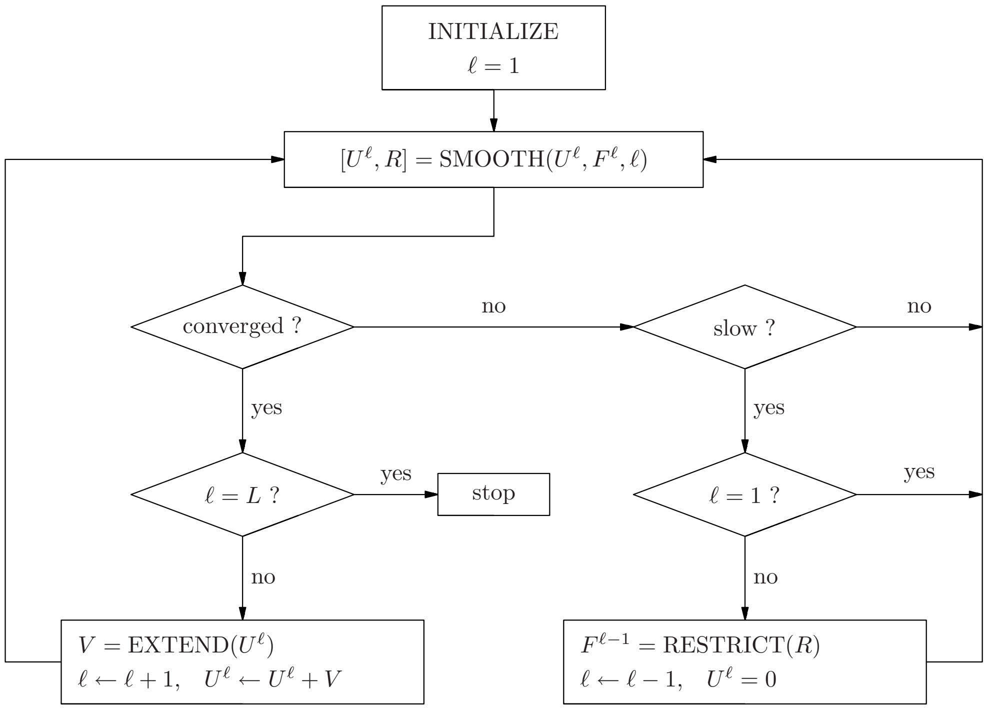
INITIALIZE
$\ell = 1$
$[U^\ell, R] = \mathrm{SMOOTH}(U^\ell, F^\ell, \ell)$
converged ?
no
slow ?
no
yes
yes
$\ell = L$?
yes
stop
$\ell = 1$?
yes
no
no
$V = \mathrm{EXTEND}(U^\ell)$
$\ell \leftarrow \ell + 1, \quad U^\ell \leftarrow U^\ell + V$
$F^{\ell-1} = \mathrm{RESTRICT}(R)$
$\ell \leftarrow \ell - 1, \quad U^\ell = 0$

Figure 3: "Cycle C" [8]: Dynamic multigrid iteration for linear elliptic problems

In accordance with our convention for the coefficients U, the Ritz-Galerkin matrix G is a four-dimensional array; the row and column indices refer to the grid positions $(j_1, j_2), (k_1, k_2)$ of the b-splines. As a consequence, the matrix multiplication $U \mapsto GU$ has to be interpreted in a generalized sense, but is of course defined in the usual way:

$$(GU)_{(j_1,j_2)} = \sum_{k_1=1}^{M+n} \sum_{k_2=1}^{M+n} g_{(j_1,j_2),(k_1,k_2)}\, u_{(k_1,k_2)}\,.$$

Keeping irrelevant elements with no support in D results in zero rows and columns of the Ritz-Galerkin matrix. To preserve the positive definiteness of the linear system, we set diagonal entries corresponding to irrelevant indices equal to 1. Since corresponding entries $f_j = \int_D B_j f,\ j \not\sim D$, of the right side are 0, this leads to the proper values $u_j = 0$ for the Ritz-Galerkin solution.

4 Dynamic Multigrid Iteration

There are numerous variants of multigrid algorithms. Which one to select, depends of course on the application and, to some extent, also on preferences of the programmer. Our personal favorite for linear problems is the dynamic iteration proposed by Brandt in his fundamental article [8] and (essentially) restated in Figure 3 for the convenience of the reader.

The multigrid strategy is based on the fact that the discretizations

$$G^\ell U^\ell = F^\ell$$

on a sequence of grids (referred to by the superscript ℓ) with grid widths

$$h^\ell = H\, 2^{1-\ell}, \quad \ell = 1, \dots, L\,,$$

are closely related, since they correspond to the same continuous problem. The dynamic multigrid algorithm exploits this in two ways.
(i) Initial approximations: First, for all levels ℓ one generates the matrices G^ℓ and right sides F^ℓ and sets $U^\ell = 0$ (subroutine `INITIALIZE`). Then, starting on the coarsest grid ($\ell = 1$), a sufficiently accurate Ritz-Galerkin solution $u^\ell = \sum_k u_k^\ell B_k^\ell$ is used as an initial approximation on the next finer grid. To this end, u^ℓ is expressed as linear combination of the refined elements (subroutine `EXTEND`),

$$u^\ell = \sum_k v_k B_k^{\ell+1}\,,$$

and used as initial guess for $u^{\ell+1}$:

$$U^{\ell+1} \leftarrow U^{\ell+1} + V, \quad V = (v_k)\,,$$

recalling that $U^{\ell+1} = 0$ initially.
(ii) Main iteration: To improve the accuracy of U^ℓ, a classical scheme such as the Jacobi or Gauß-Seidel method is used as basic iteration. Typically, a fixed number α of steps is performed (subroutine `SMOOTH`) before a convergence test is made. As is well known, the basic classical iterations smooth the residual

$$R = F^\ell - G^\ell U^\ell\,,$$

i.e., oscillating error components are rapidly damped. Hence, if the convergence becomes slow (otherwise, no action is necessary), we can compute a correction $V \approx (G^\ell)^{-1} R$, to be added to the current approximation U^ℓ, on the coarser grid. To this end, the residual is transferred to the coarser grid (subroutine `RESTRICT`), where the dominant slowly varying components can be adequately represented:

$$F^{\ell-1} \leftarrow R\,.$$

For the iterative solution of the residual equation $G^{\ell-1}U^{\ell-1} = F^{\ell-1}$, which presumably determines a small correction, $U^{\ell-1} = 0$ is the appropriate initial guess. Once sufficient accuracy is obtained, the current approximation is improved by transferring the coarse grid correction to the finer grid (subroutine `EXTEND`) and adding it to U^ℓ:

$$V \leftarrow U^{\ell-1}, \quad U^\ell \leftarrow U^\ell + V\,.$$

This strategy is applied recursively. On the coarsest grid, where no transfer of the residual is possible, the iteration continues until the accuracy criterion is met.

Although the Gauß-Seidel scheme yields the best convergence rates, we prefer the Jacobi iteration. The less efficient smoothing is outweighed by the ease of implementation and, more important, vectorization, necessary in particular for large scale applications.

The implementation of the dynamic iteration based on Jacobi's scheme requires subroutines for assembling the Ritz-Galerkin matrices, matrix/vector multiplication in sparse format, and grid transfer. Essentially, this is all! Each of these multigrid components will be described in detail in the next sections. Here, we comment briefly on the general algorithmic parameters, following the discussion in [8].

The convergence rate on a particular grid can be estimated via the quotient of the residual norms, which are computed in conjunction with the Jacobi steps. Convergence is slow if

$$\|R_{\text{new}}\|/\|R_{\text{old}}\| > \beta$$

with β slightly less than 1.

To judge the accuracy of an approximation on grid level ℓ, we compare the norms of the residual R and the right side F^ℓ (equal to the residual for the initial guess $U^\ell = 0$). The iteration has converged if

$$\|R\| \leq \text{tol}\, \|F^\ell\| \, .$$

The constant tol depends on the level and the phase of the iteration. On a grid, where the solution to the boundary value problem is currently approximated, the relative accuracy should be proportional to the truncation error:

$$\text{tol} = \gamma_1 \, (h^\ell)^{n+1}$$

with γ_1 rather small (n equals the degree). On grids, where corrections are currently computed, a moderate reduction of the size of the residual is sufficient:

$$\text{tol} = \gamma_2$$

with γ_2 significantly less than 1, but independent of the grid width h^ℓ.

According to this strategy, initially tol $= \gamma_1 \, (h^\ell)^{n+1}$ on all grids. After having computed an accurate numerical solution on grid ℓ and passed as initial approximation to grid $\ell + 1$, we set tol $= \gamma_2$ on grid ℓ. This was done already on the coarser grids $1, \ldots, \ell - 1$, where numerical solutions were previously generated. On the finest grid, tol $= \gamma_1 \, (h^\ell)^{n+1}$ during the entire dynamic multigrid iteration.

From our practical experience, good choices of the parameters are $\alpha = 5$, $\beta = 0.8$, $\gamma_1 = 0.01$, and $\gamma_2 = 0.2$. The optimal values can vary, depending on the problem and the type of basis functions (degree, weight function, parametrization, etc.). However, as has been pointed out in [8], generally, dynamic multigrid algorithms are fairly robust with respect to small changes in the parameters. Clearly, increasing α, β and decreasing tol improve stability of the iteration while requiring more operations per multigrid cycle.

5 Matrix Assembly and Initialization

As for conventional finite element methods, Ritz-Galerkin systems $GU = F$ for weighted isogeometric b-spline elements are computed with the aid of numerical integration over the parameter domain. The standard procedure is easily described (cf., e.g., [17, 12]). We review the principal ideas mainly for the convenience of the reader and to introduce the data structure for the multigrid algorithm. For the sake of brevity, we only consider the assembly of the matrix G. The computation of the entries of the right side F is simpler and completely analogous.

First, we observe that

$$\begin{aligned}\operatorname{grad}_x B_k(x) &= (\operatorname{grad}_x w(x))\, b_k(\Phi^{-1}(x)) + w(x)\,(\operatorname{grad}_x b_k(\Phi^{-1}(x)) \\ &= (\operatorname{grad}_x w)(\Phi(\xi))\, b_k(\xi) + w(\Phi(\xi))\,(\operatorname{grad}_\xi b_k(\xi)\Phi'(\xi)^{-1})\end{aligned}$$

by the chain rule, where Φ' denotes the Jacobi matrix of the parametrization. Abbreviating the expression for $\operatorname{grad}_x B_k$ by $p(k,\xi)$ and denoting by $\Omega = \Phi^{-1}(D)$ the portion of Q mapped onto D by Φ, the nontrivial entries of the Ritz-Galerkin matrix equal

$$g_{j,k} = \int_\Omega \underbrace{\big(p(j,\xi)p(k,\xi) + w(\Phi(\xi))^2 b_j(\xi)b_k(\xi)\big)\,|\det \Phi'(\xi)|}_{q(j,k,\xi)}\, d\xi, \quad j,k \sim D\,,$$

by the rule for transformation of multiple integrals.

Second, we note that G is most easily assembled by summing the contributions from each grid cell Q_m:

$$\int_\Omega q(j,k,\xi)\, d\xi = \sum_{m_1=1}^{M} \sum_{m_2=1}^{M} \int_{Q_m\cap\Omega} q(j,k,\xi)\, d\xi\,,$$

where only those summands are relevant for which Q_m lies in the common supports of b_j and b_k, i.e.,

$$m_\nu \le j_\nu, k_\nu \le m_\nu + n \quad (\nu = 1,2)\,.$$

The integrals over the grid cell intersections are approximated with the aid of an appropriate quadrature rule, as is illustrated in Figure 4:

$$\int_{Q_m\cap\Omega} q(j,k,\xi)\, d\xi \approx \sum_i r_m(i) q(j,k,\eta_m(i))\,.$$

If $Q_m \subset \Omega$, a standard tensor product Gauß formula is applicable. However, if the preimage of Γ intersects Q_m, we have to use an appropriate partition in order to preserve the high accuracy of the b-spline method (cf. [17] for details). The weights r_m and the nodes η_m involve a combination of Gauß formulas in such cases. Clearly, if $Q_m \cap \Omega = \emptyset$, we set $r_m(i) = 0$.

The matrix entry $g_{j,k}$ is nonzero only if the support of the b-splines b_j and b_k overlap, i.e., if

$$s_\nu = k_\nu - j_\nu \in \{-n, \ldots, n\}, \quad \nu = 1,2\,.$$

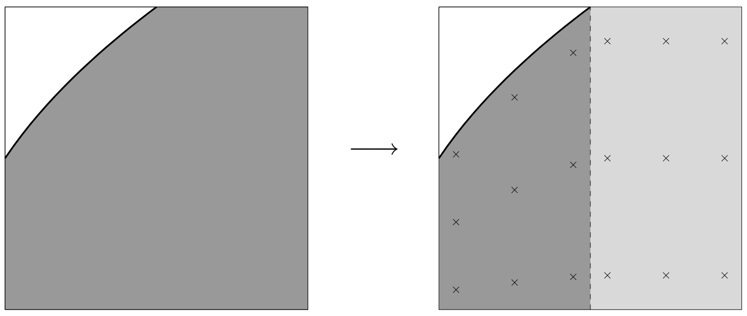

Figure 4: Gauß points for cell integrals

Hence, we can store G in sparse format, i.e., in a condensed array A of dimension $(M+n)\times(M+n)\times(2n+1)\times(2n+1)$:

$$g_{j,k} = a_{j,s}, \quad 1 \le j_\nu, k_\nu \le M+n,\ -n \le s_\nu = k_\nu - j_\nu \le n\,, \tag{1}$$

for $\nu = 1,2$. The (generalized) columns of A, $a_{(\cdot,\cdot),s}$, contain the diagonals of G with the index s corresponding to the offset from the main diagonal. We allow negative indices s_ν to keep the symmetry, which is convenient in formulas. In programs, we have to shift the index appropriately, i.e.,

$$a^{\text{program}}_{j,s} = a^{\text{theory}}_{j,s+(n+1,n+1)}\,,$$

which is not a serious inconsistency.

We note that the entries $a_{j,s}$ with $k_\nu = j_\nu + s_\nu \notin \{1,\ldots,M+n\}$ for $\nu = 1$ or $\nu = 2$ do not correspond to matrix elements. This is due to the slightly different lengths of the diagonals of G. Defined as 0, these elements do not affect the result in matrix operations, which is convenient in programs. Also, with this convention, there is no need for specifying the proper range of the indices s_ν. Similarly, we set elements of a vector to 0 if their index is out of range.

After these preliminary considerations, the assembly subroutine for the Ritz-Galerkin matrix has a very simple form. It uses auxiliary arrays

```
WEIGHTS{m1,m2}(:),  POINTS{m1,m2}(:,1:2) ,
```

which store the integration weights $r_m(i)$ and nodes $(\eta_m(i,1),\eta_m(i,2))$ for the grid cells Q_m and a subroutine

```
VALUES(j1,j2,k1,k2,P) ,
```

which evaluates the integrand $q(j,k,\cdot)$ at the points `P(i,1:2)`. In the program listed below, for each grid cell Q_m, the contributions of element combinations B_j, B_k with overlapping supports are computed and successively added in the appropriate positions of A (the offset indices s_ν shifted by $n+1$ range from 1 to $2n+1$). In a final loop, the diagonal entries corresponding to irrelevant indices $j \not\sim D$ are modified.

```
function A = SYSTEM(WEIGHTS,POINTS,VALUES,n)
% assembly of a Ritz-Galerkin matrix

M = size(WEIGHTS,1);
A = zeros(M+n,M+n,2*n+1,2*n+1);
for m1=1:M, for m2=1:M
    for j1=m1:m1+n, for j2=m2:m2+n
        for k1=m1:m1+n, for k2=m2:m2+n
            s1 = k1-j1+n+1; s2 = k2-j2+n+1;
            V = VALUES(j1,j2,k1,k2,POINTS{m1,m2});
            A(j1,j2,s1,s2) = A(j1,j2,s1,s2) + WEIGHTS{m1,m2}'*V;
        end, end
    end, end
end, end
for j1=1:M+n, for j2=1:M+n
    if A(j1,j2,n+1,n+1)==0
        A(j1,j2,n+1,n+1)=1;
    end
end, end
```

The simplicity of the code is deceptive. Of course, the reader realizes that we hide the geometric complexity in the arrays `WEIGHTS` and `POINTS` and the evaluation of the weighted isogeometric b-splines in the subroutine `VALUES`. Let us comment on these aspects in some more detail.

The parameters for numerical integration are computed in a preprocessing step. First, the arrays `WEIGHTS{m1,m2}(:)` and `POINTS{m1,m2}(:,1:2)` are initialized with standard Gauß parameters for squares. For cells outside the domain, the weights are set to 0. Then, for those m, for which the grid cell Q_m intersects the boundary of Ω, the parameters are modified. Depending on the particular representation of the weight function w and the complexity of the boundary of $Q_m \cap \Omega$, this can be time-consuming. But, there are at most $O(h^{-1})$ such boundary cells, many of which can be handled with just one subdivision. Hence, generating the integration parameters requires only a small portion of the overall computing time.

The function `VALUES` describing the Ritz-Galerkin integrand just involves the evaluation of the weight function w, the b-splines b_k, and the parametrization Φ. This is straightforward and fast since no inversion of Φ is necessary.

In the initialization phase of the multigrid algorithm, the matrices G^ℓ and right sides F^ℓ have to be assembled for all grid levels $\ell = 1, \ldots, L$. To this end, it is important to note that numerical integration, as described above, is necessary only for the finest grid level L. The systems for coarser grids can be generated from G^L, F^L with the aid of grid transfer operations, as will be discussed in Section 8.

To stabilize and accelerate the iteration, we precondition the Ritz-Galerkin system by using symmetric diagonal scaling, i.e.,

$$G \to \widehat{G} = \operatorname{diag}(G)^{-1/2}\, G\, \operatorname{diag}(G)^{-1/2}\,.$$

The preconditioned matrices in sparse format, the square roots of the original diagonals, and the right sides are stored in arrays

$$\texttt{A}\{\ell\},\ \texttt{DA}\{\ell\},\ \texttt{F}\{\ell\},\quad \ell = 1,\ldots,L;$$

the cell arrays `A`, `DA`, and `F` are global variables. It is also convenient to use a global structure `PAR` for the algorithmic parameters:

`PAR.n`, `PAR.L`, `PAR.H`, `PAR.alpha`, `PAR.beta`, `PAR.gamma(1:2)`, `PAR.rho(1:L)` .

They contain the degree, the number of levels, the maximal grid width, the number of Jacobi steps, the smoothing rate, the relative accuracies, and the maximum norms of the preconditioned Ritz-Galerkin matrices.

6 Smoothing Iteration

In this section we describe the implementation of the Jacobi scheme, which is one of the basic components of the multigrid iteration. Using familiar concepts for banded matrices, the b-spline data structure can be exploited in a straightforward fashion.

We begin by describing the multiplication with the preconditioned Ritz-Galerkin matrix $\widehat{G}$, an operation needed for computing the residual, hence in particular also for implementing a Jacobi step. Recalling that $\widehat{G}$ is stored in sparse format in the array A, as described in (1), the multiplication $U \to V = \widehat{G}U$ has the explicit form

$$v_j = \sum_{s_1=-n}^{n} \sum_{s_2=-n}^{n} a_{j,s}\, u_{j+s}, \quad 1 \le j_\nu \le M+n,\ \nu = 1,2\,. \tag{2}$$

We recall our convention that $u_k = 0$ for $k = j+s \notin \{1,\ldots,M+n\}^2$. Such elements are multiplied by entries $a_{j,s}$, $s = k-j$, which do not represent matrix elements of $\widehat{G}$ and have been set to 0, too.

The extremal index pairs occurring are

$$k = (1-n, 1-n), \quad k = (M+2n, M+2n)\,.$$

Hence, to implement the formula (2), we pad the $(M+n)\times(M+n)$ array U with n layers of zeros on both sides. Denoting the enlarged array by U°, i.e., setting

$$u_{k_1-n,k_2-n} = u^\circ_{k_1,k_2}\,,$$

the matrix vector multiplication can be computed with the following MATLAB program.

```
function V = MULT(U,l)
% multiplication with a Ritz-Galerkin matrix

global A, PAR
```

```
Al = A{l}; n = PAR.n; M = size(U,1)-n;
J = 1:M+n;
U0 = zeros(M+3*n,M+3*n); U0(n+J,n+J) = U;
V = zeros(M+n,M+n);
for s1=1:2*n+1, for s2=1:2*n+1
   V = V + Al(:,:,s1,s2) .* U0(J+s1-1,J+s2-1);
end, end
```

We note the perfect vectorization: $(2n+1)^2$ additions and pointwise multiplications of $(M+n)\times(M+n)$ dimensional arrays.

We now consider a Jacobi smoothing step for solving $GU = F$. As mentioned before, we use symmetric diagonal preconditioning, i.e., we solve

$$\widehat{G}\widehat{U} = \widehat{F} \quad \Leftrightarrow \quad \left(\mathrm{diag}(G)^{-1/2}\, G\; \mathrm{diag}(G)^{-1/2}\right)\, \left(\mathrm{diag}(G)^{1/2}\, U\right) = \left(\mathrm{diag}(G)^{-1/2}\, F\right) .$$

Then, one step of the iteration has the form

$$\widehat{U} \leftarrow \widehat{U} + \varrho^{-1}(\widehat{F} - \widehat{G}\widehat{U})$$

with $\varrho = \max_j(\sum_k |\hat{g}_{j,k}|)$ equal to the maximum norm of $\widehat{G}$. With the aid of the program `MULT` for multiplying with the preconditioned Ritz-Galerkin matrix, the Jacobi smoothing is easily implemented.

```
function [U,R] = SMOOTH(U,F,l)
% Jacobi smoothing steps

global A, DA, PAR

Al = Al; DAl = DAl; RHOl = PAR.rho(l);
U = U.*DAl; F = F./DAl;
for i = 1:PAR.alpha
    U = U + (F-MULT(U,l))/RHOl;
end
R = (F - MULT(U,l)).*DAl;
U = U./DAl;
```

We note that, while the Ritz-Galerkin matrix is stored in preconditioned form in the cell array A, the preconditioning has to be applied to the vectors U and F before the Jacobi steps. After the loop, the diagonal scaling is reversed.

7 Grid Transfer

The modules `EXTEND` and `RESTRICT` for changing between grid levels are based on the formula for subdividing a b-spline b_k [25, 10, 5], which is illustrated in Figure 5. Accordingly, we can express a linear combination of b-splines $\tilde{b}_k$ on a coarser grid in

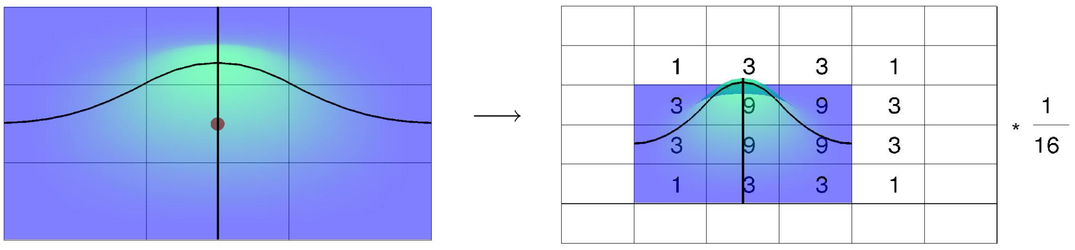

Figure 5: Subdivision of a bi-quadratic b-spline

terms of the fine grid b-splines b_j; this is of course also possible for the corresponding weighted isogeometric elements:

$$\widetilde{B}_k = \sum_j p_{j,k} B_j \quad \Longrightarrow \quad \sum_{k_1=1}^{M/2+n} \sum_{k_2=1}^{M/2+n} v_k \widetilde{B}_k = \sum_{j_1=1}^{M+n} \sum_{j_2=1}^{M+n} u_j B_j, \quad U = PV\,, \tag{3}$$

where the sums are taken over all b-splines which are relevant for the parameter domain $Q = [0,1]^2$.

For univariate splines, the coefficient vector U can be computed from V by two extremely simple operations:
(E1): The elements of V are doubled and divided by 2^n, i.e.,

$$U \leftarrow 2^{-n}\,(v_1, v_1, v_2, v_2, \ldots)^t\,.$$

(E2): A simultaneous (scaled by 2) average

$$U \leftarrow (u_1 + u_2, u_2 + u_3, \ldots)^t$$

is repeated n times.

The first step doubles the size of the coefficient vector. In the second step, each average shrinks the size of U by 1. Accordingly, the dimension changes from $M/2+n$ to $M+n$, in agreement with equation (3).

For bivariate b-splines, the subdivision operations are applied separately in each coordinate direction. We illustrate the effect of the operations (E1) and (E2) for degree $n = 1$:

$$V \overset{E_1}{\to} \frac{1}{4} \begin{pmatrix} v_{1,1} & v_{1,1} & v_{1,2} & v_{1,2} & \cdots \\ v_{1,1} & v_{1,1} & v_{1,2} & v_{1,2} & \\ v_{2,1} & v_{2,1} & v_{2,2} & v_{2,2} & \\ v_{2,1} & v_{2,1} & v_{2,2} & v_{2,2} & \\ \vdots & & & & \ddots \end{pmatrix} \overset{E_2}{\to} \begin{pmatrix} v_{1,1} & \frac{v_{1,1}+v_{1,2}}{2} & v_{1,2} & \cdots \\ \frac{v_{1,1}+v_{2,1}}{2} & \frac{v_{1,1}+v_{1,2}+v_{2,1}+v_{2,2}}{4} & \frac{v_{1,2}+v_{2,2}}{2} & \\ v_{2,1} & \frac{v_{2,1}+v_{2,2}}{2} & v_{2,2} & \\ \vdots & & \ddots & \end{pmatrix} \to U\,.$$

The simplicity of the resulting subdivision algorithm, reflected by the MATLAB program below, is striking: step (E1) makes four copies of each element, step (E2), repeated n times, forms averages of four neighbors.

```
function U = EXTEND(V)
% extension to a finer grid

global PAR
n = PAR.n;
K = ceil([1/2 :  1/2 :  size(V,1)]);
U = V(K,K)/4^n;
for i=1:n
    U = U(1:end-1,:)  + U(2:end,:);
    U = U(:,1:end-1) + U(:,2:end);
end
```

Step (E1) is implemented by creating an index vector $K = (1, 1, 2, 2, \ldots)$. The loop, corresponding to step (E2), computes the averages simultaneously for the entire array - a perfect vector operation.

As is common practice for finite element discretizations, we use the transpose of the extension matrix for the restriction to a coarser grid. To this end, we need the matrix representation P of the subdivision operations (E1) and (E2). Considering univariate b-splines first,

$$U = PV, \quad P = \underbrace{P^n \cdots P^1}_{\text{step (E2)}} \; \underbrace{2^{-n}\, P^0}_{\text{step (E1)}},$$

where

$$p^0_{j,k} = \delta_{j-2k+1} + \delta_{j-2k}, \quad p^1_{jk} = \cdots = p^n_{j,k} = \delta_{j-k} + \delta_{j-k+1} \tag{4}$$

with the symbol δ defined as

$$\delta_k = \begin{cases} 1 & \text{for } k = 0 \\ 0 & \text{otherwise} \end{cases} .$$

The superscripts $1, \ldots, n$ are necessary merely since the dimension changes; each (scaled) average reduces the vector length by 1. It may be helpful to give a concrete example. Below we list the matrices for univariate subdivision of a linear combination of quadratic b-splines $\sum_{k=1}^{3} v_k \tilde{b}_k$:

$$P^2 = \begin{pmatrix} 1 & 1 & & & 0 \\ & 1 & 1 & & \\ & & 1 & 1 & \\ 0 & & & 1 & 1 \end{pmatrix}, \quad P^1 = \begin{pmatrix} 1 & 1 & & & & 0 \\ & 1 & 1 & & & \\ & & 1 & 1 & & \\ & & & 1 & 1 & \\ 0 & & & & 1 & 1 \end{pmatrix}, \quad P^0 = \begin{pmatrix} 1 & & 0 \\ 1 & & \\ & 1 & \\ & 1 & \\ & & 1 \\ 0 & & 1 \end{pmatrix}.$$

Here, $M = 2$, and the vectors V and U have dimensions $M/2 + 2 = 3$ and $M + 2 = 4$, respectively.

Staying in the univariate setting, the effect of the transposition is easily determined. Writing

$$P^t = 2^{-n}\,(P^0)^t\,(P^1)^t \cdots (P^n)^t\,,$$

we consider the two types of factors in turn. Referring to (4),

$$V = (P^0)^t U \Leftrightarrow v_j = \sum_k (\delta_{k-2j+1} + \delta_{k-2j})\, u_k = u_{2j-1} + u_{2j}\,,$$

i.e., multiplication by P^0 just adds two adjacent vector elements. Again, by definition (4), for $\mu = 1, \ldots, n$,

$$V = (P^\mu)^t U \Leftrightarrow v_j = \sum_k (\delta_{k-j} + \delta_{k-j+1})\, u_k = u_j + u_{j-1}\,,$$

where we recall that $u_j = 0$ for indices out of range. Since the dimension J of V is by 1 larger than the dimension of U, $u_0 = 0 = u_J$. With this convention, the above matrix/vector multiplication is identical to the operation (E2) if we append a zero entry to U on both sides. Summarizing, the (univariate) restriction V of a vector U is computed by the following two operations:
(R2): A simultaneous (scaled) average of an extended vector,

$$U \leftarrow (0 + u_1, u_1 + u_2, \ldots, u_{K-1} + u_K, u_K + 0)^t\,,$$

with K the current length of U, is repeated n times.
(R1): Pairs of consecutive elements of U are added and divided by 2^n:

$$V \leftarrow 2^{-n}\,(u_1 + u_2, u_3 + u_4, \ldots)^t\,,$$

halving the length of the vector.

The procedure generalizes to two dimensions in a by now familiar fashion. The operations (R2) and (R1) are performed separately in each variable.

(R2$_{\mathrm{2d}}$): For $k_\nu = 1, \ldots, K+1$,

$$u_k \leftarrow u_{(k_1-1,k_2)} + u_{(k_1,k_2)}, \quad u_k \leftarrow u_{(k_1,k_2-1)} + u_{(k_1,k_2)}\,,$$

where K is the current dimension of the input vector U, and we recall that elements with indices out of range (here $k_\nu = 0$ and $k_\nu = K+1$) are set to 0.
(R1$_{\mathrm{2d}}$): $V \leftarrow 4^{-n}\, U$ and

$$v_j \leftarrow v_{(2j_1-1,j_2)} + v_{(2j_1,j_2)}, \quad v_j \leftarrow v_{(j_1,2j_2-1)} + v_{(j_1,2j_2)}\,.$$

This leads to the following MATLAB program.

```
function V = RESTRICT(U)
% restriction to a coarser grid

global PAR
n = PAR.n; M = size(U,1);
V = zeros(M+2*n,M+2*n);
V(1+n:M+n,1+n:M+n) = U/4^n;
```

```
for i=1:n
    V = V(1:end-1,:)  + V(2:end,:);
    V = V(:,1:end-1) + V(:,2:end);
end
V = V(1:2:end,:)  + V(2:2:end,:);
V = V(:,1:2:end) + V(:,2:2:end);
```

We notice a minor modification compared to the theoretical descriptions of steps ($\mathrm{R2}_{\mathrm{2d}}$) and ($\mathrm{R1}_{\mathrm{2d}}$). Instead of padding with one layer of zeros in each of the n averaging steps, we pad the input array immediately by n layers of zeros. Programmed in this way, the loop corresponding to ($\mathrm{R2}_{\mathrm{2d}}$) is identical to the implementation of (E2). Essentially, the only major difference to the program `EXTEND` is the realization of ($\mathrm{R1}_{\mathrm{2d}}$) in the last two statements.

The short implementations of the extension and restriction modules reflect the elegance of subdivision algorithms, which have become powerful tools, in particular in computer graphics (cf., e.g., [27] for a comprehensive treatment of the beautiful mathematical theory in a much more general context).

8 Coarse Grid Ritz-Galerkin Matrices

The Ritz-Galerkin matrices on coarse grids can, of course, be assembled as described in Section 5. New weights and nodes do not have to be generated. Instead, the parameters for four small squares, which form a large square, can be combined. While proceeding in this way is reasonably efficient, a more elegant alternative is to use subdivision. As will be explained below, the Ritz-Galerkin matrix $\widetilde{G}$ on a coarser grid (before preconditioning) can be obtained from the fine grid matrix G by a simple averaging procedure.

We use the fact that the entries $\tilde{g}_{j,k}$ depend bilinearly on the weighted isogeometric elements, which we indicate by writing

$$\tilde{g}_{j,k} = \pi(\widetilde{B}_j, \widetilde{B}_k)\,.$$

Substituting the subdivision formula (3), $\widetilde{B}_i = \sum_\mu p_{\mu,i} B_\mu$,

$$\tilde{g}_{j,k} = \sum_\sigma \sum_\tau p_{\sigma,j}\, \underbrace{\pi(B_\sigma, B_\tau)}_{g_{\sigma,\tau}}\, p_{\tau,k} \quad \Leftrightarrow \quad \widetilde{G} = P^t G P\,.$$

Hence, the transformation has a very simple form. Multiplying by P^t from the left (P from the right) means applying the operations ($\mathrm{R2}_{\mathrm{2d}}$), ($\mathrm{R1}_{\mathrm{2d}}$) simultaneously to all columns (rows) of G. A slight complication is the diagonal storage format for the Ritz-Galerkin matrices which prevents us from using just the program `RESTRICT`. However, the implementation of the $n+1$ successive modifications of G (n applications of operation ($\mathrm{R2}_{\mathrm{2d}}$) and one application of operation ($\mathrm{R1}_{\mathrm{2d}}$)) is similar. We discuss each step in turn.

($\mathrm{R2}_{\mathrm{matrix}}$): Recalling the relation (1) between an input matrix G and the corresponding array A, the row operations have the form

$$\begin{array}{ccccccc} a_{j,s} &=& g_{j,k} &\leftarrow& g_{j,(k_1-1,k_2)} + g_{j,(k_1,k_2)} &=& a_{j,(s_1-1,s_2)} + a_{j,(s_1,s_2)} \\ a_{j,s} &=& g_{j,k} &\leftarrow& g_{j,(k_1,k_2-1)} + g_{j,(k_1,k_2)} &=& a_{j,(s_1,s_2-1)} + a_{j,(s_1,s_2)}\,, \end{array} \tag{5}$$

where $s_\nu = k_\nu - j_\nu$. Similarly, the column operations lead to the modifications

$$\begin{array}{ccccccc} a_{j,s} &=& g_{j,k} &\leftarrow& g_{(j_1-1,j_2),k} + g_{(j_1,j_2),k} &=& a_{(j_1-1,j_2),(s_1+1,s_2)} + a_{(j_1,j_2),(s_1,s_2)} \\ a_{j,s} &=& g_{j,k} &\leftarrow& g_{(j_1,j_2-1),k} + g_{(j_1,j_2),k} &=& a_{(j_1,j_2-1),(s_1,s_2+1)} + a_{(j_1,j_2),(s_1,s_2)}\,. \end{array} \tag{6}$$

We recall that all vectors are padded with zeros at both ends before performing the averaging operations. Hence, each dimension of G increases by one and the number of diagonals (the third and fourth dimension of the array A) by two.
($\mathrm{R1}_{\mathrm{matrix}}$): The operations are very similar. Scaled averages are formed, but only for every second pair. Since this halves each dimension of an input array G, treating each component separately, as done in ($\mathrm{R2}_{\mathrm{matrix}}$), does not fit with the diagonal storage pattern. Hence, we have to combine the row and column operations for each coordinate direction. For the first coordinate, this yields

$$\begin{array}{ccccl} a_{j,s} &=& g_{j,k} &\leftarrow& g_{(2j_1-1,j_2),(2k_1-1,k_2)} + g_{(2j_1-1,j_2),(2k_1,k_2)} + g_{(2j_1,j_2),(2k_1-1,k_2)} + g_{(2j_1,j_2),(2k_1,k_2)} \\ && &=& a_{(2j_1-1,j_2),(2s_1,s_2)} + a_{(2j_1-1,j_2),(2s_1+1,s_2)} + a_{(2j_1,j_2),(2s_1-1,s_2)} + a_{(2j,j_2),(2s_1,s_2)}\,. \end{array} \tag{7}$$

Clearly, the operations are identical for the second coordinate, just applied with respect to the second and fourth index of A.

Before giving an implementation of the above difference equations, let us comment on the dimensions $J \times J \times S \times S$ of the array A modified by the algorithm. From the definition of the Ritz-Galerkin matrices we recall that $1 \le j_\nu \le M/2 + n$, $|s_\nu| \le n$, for the output array. Hence, on the right side of (7), $j_\nu \in \{1, \ldots, M + 2n\}$ and $s_\nu \in \{-2n-1, \ldots, 2n+1\}$. Therefore, before step ($\mathrm{R1}_{\mathrm{matrix}}$),

$$J = M + 2n, \quad S = 4n + 3\,.$$

In equations (5), (6), the range of the first and second index of the array A is by 1 larger on the right side, the range of the third and fourth index by 2 (the indices $s_\nu - 1, s_\nu, s_\nu + 1$ appear). Hence, proceeding backwards, the dimensions increase according to

$$J \to J + 1, \quad S \to S + 2\,.$$

Since the operations ($\mathrm{R2}_{\mathrm{matrix}}$) are repeated n times, initially

$$J = (M + 2n) + n, \quad S = (4n + 3) + 2n\,.$$

Comparing these values with the dimension $(M + n)^2 \times (2n + 1)^2$ of the input array, and recalling our convention that entries out of range are set to 0, we must pad with $2 \times n$ layers of zeros in dimensions $1, 2$ and with $2 \times (2n + 1)$ layers of zeros in dimensions $3, 4$.

```
function A0 = MATRIX(A1)
% Ritz-Galerkin matrix for a coarser grid

global PAR
n = PAR.n; M = size(A1,1)-n;
A0 = zeros(M+2*[n,n,2*n+1,2*n+1]);
A0(1+n:end-n,1+n:end-n,2+2*n:end-2*n-1,2+2*n:end-2*n-1) = A1/16^n;
for i = 1:n
    A0 = A0(:,:,1:end-1,:)  + A0(:,:,2:end,:);
    A0 = A0(:,:,:,1:end-1) + A0(:,:,:,2:end);
    A0 = A0(1:end-1,:,2:end,:)  + A0(2:end,:,1:end-1,:);
    A0 = A0(:,1:end-1,:,2:end) + A0(:,2:end,:,1:end-1);
end
M = size(A0); J = 1:2:M(1)-1; S = [2:2:M(3)-1];
A0 = A0(J,:,S,:)  + A0(J,:,S+1,:)  + A0(J+1,:,S-1,:)  + A0(J+1,:,S,:);
A0 = A0(:,J,:,S) + A0(:,J,:,S+1) + A0(:,J+1,:,S-1) + A0(:,J+1,:,S);
```

There seem a little bit too many instructions in this program. A shorter version is possible. For example, using an alternative version of ($\mathrm{R2}_{\mathrm{matrix}}$),

$$g_{j,k} \leftarrow \sum_{r,t\in I} g_{j-r,k-t}, \quad I = \{0,1\}^2\,,$$

we can replace the statements in the loop by the following code segment.

```
A = zeros(size(A0)-[1;1;2;2]);
for r1=0:1, for r2=0:1, for t1=0:1, for t2=0:1
    A = A + A0([2:end]-r1,[2:end]-r2,[2:end-1]+r1-t1,[2:end-1]+r2-t2);
end, end, end, end
A0 = A;
```

Similar changes can be made also for the last two statements of `MATRIX` and in other programs. The possible reductions are listed in Figure 6 which in brackets shows the minimal number of floating point instructions achievable by introducing additional loops. However, fewer loops usually provide a slightly more efficient implementation. For the above example, the four operations of the original program segment compare favorably with the 2^4-fold repetition of the statement within the four loops.

9 Convergence

Since the classical work by Fedorenko [14], Bakhvalov [2], Bank and Dupont [3], numerous proofs of grid independent convergence rates have been given (cf., in particular [7] for a general framework). By now, the arguments are standard, and apply

	SYSTEM	MULT	SMOOTH	EXTEND	RESTRICT	MATRIX
.*, ./	1	1	4	0	0	0
$\pm$	1	1	3	2 [1]	4 [2]	10 [2]

Figure 6: Actual and minimal (in brackets) number of floating point instructions of MATLAB programs for key multigrid components

also for some b-spline-based methods [21]. For a second order elliptic boundary value problem, they rely usually on the following properties of finite element basis functions B_k.

- Stability: The L_2-norm of approximations is comparable to the 2-norm of the coefficients:
$$\left\|\sum_k u_k B_k\right\|_{L^2(D)} \asymp h^{d/2}\left(\sum_k |u_k|^2\right)^{1/2},$$
where d is the dimension ($d = 2$ throughout this article).
- Bernstein Inequality: Differentiation increases the norm of approximations $u^h = \sum_k u_k B_k$ at most by a factor h^{-1}, i.e.
$$\left\|u^h\right\|_{H^1(D)} \preceq h^{-1}\left\|u^h\right\|_{L^2(D)}.$$
- Jackson Inequality: There exists a projector P^h onto the finite element subspace which is bounded in $L^2(D)$ and approximates with optimal order, i.e.
$$\left\|u - P^h u\right\|_{H^\ell(D)} \preceq h^{n+1-\ell}\|u\|_{H^2(D)}, \quad \ell = 0, 1.$$

In the above inequalities, the symbols $\preceq, \asymp$ stand for inequalities in one or both directions with constants which do not depend on u, U, or h.

The first property, stability, is crucial for all the proofs given so far. However, it is not valid for the weighted isogeometric elements which were used in this article. Due to the curved boundary Γ, there can exist b-splines with very small support in D, and this leads to an ill-conditioned basis. The problem can be overcome with an extension procedure, which is part of the construction of web-splines [20], and should easily be adapted to the present situation. In two variables, extension seems to be not even necessary. Reif and Mößner [26] showed that in many cases simple diagonal scaling stabilizes b-spline bases on domains with curved boundaries. Regardless of which method is applied, the stabilized weighted isogeometric elements should have all of the above properties. We did not incorporate such stabilization measures here since practical experience has shown that stabilization seems to be essential neither for accuracy nor for the convergence of multigrid algorithms. A possible explanation is that diagonal preconditioning has a stabilizing effect for standard iterative solvers. Hence, rather than duplicating previous proofs, we conclude with an open question. Can the grid-independent convergence rate of multigrid algorithms for b-splines be proved under less stringent stability assumptions?

Conclusion

B-splines have become standard tools in approximation, modeling, and simulation, gradually replacing less sophisticated techniques. We think that b-splines will eventually play a key role for finite element methods, too. Familiar advantages of the b-spline calculus have already been successfully demonstrated in many simulations involving partial differential equations. We will not present the long list of favorable properties here and refer to [12, 17]). We just mention several key features which are apparent from the multigrid algorithms described in this article:

- regular data structure;
- simple, short, and elegant programs;
- fully vectorizable code;
- superior computational efficiency.

Clearly, our concluding statements reflect our faible for b-splines. Coming back to our remarks in the introduction, we hope to have provided some incentives to participate in the development of b-spline specific finite element software.

References

[1] G. Apaydin: *Finite Element Method with Web-splines for Electromagnetics,* VDM Verlag, Saarbrücken 2009.

[2] N. S. Bakhvalov: *On the convergence of a relaxation method with natural constraints on the elliptic operator,* USSR Comp. Math. and Math. Phys. 6 (1966), 101–135.

[3] R.E. Bank and T. Dupont: *An optimal order process for solving finite element equations,* Math. Comp. 36 (1981), 35–51.

[4] Y. Bazilevs, V.M. Calo, T.J.R. Hughes, and Y. Zhang: *Isogeometric fluid-structure interaction: Theory, algorithms and computations,* Comput. Mech. 43 (2008), 3–37.

[5] W. Böhm: *Inserting new knots into B-spline curves,* Computer-Aided Design 12 (1980), 199–201.

[6] C. de Boor: *A Practical Guide to Splines,* Springer-Verlag, New York, 1978.

[7] D. Braess, M. Dryja, and W. Hackbusch: *A multigrid method for nonconforming fe-discretizations with application to non-matching grids,* Computing 63 (1999), 1–25.

[8] A. Brandt: *Multi-level adaptive solutions to boundary value problems,* Math. Comp. 31 (1977), 333–390.

[9] W.L. Briggs, V.E. Henson, and S.F. McCormick: *A Multigrid Tutorial,* SIAM, 2000.

[10] E. Cohen, T. Lyche, and R. Riesenfeld: *Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics,* Computer Graphics Image Proc. 14 (1980), 87–111.

[11] E. Cohen, R.F. Riesenfeld, and G. Elber: *Geometric Modeling with Splines,* A K Peters, 2001.

[12] J.A. Cottrell, T.J.R. Hughes, and Y. Bazilevs: *Isogeometric Analysis,* Wiley, 2009.

[13] G.E. Farin: *Curves and Surfaces for Computer Aided Geometric Design,* Academic Press, 1988.

[14] R.P. Fedorenko: *The speed of convergence of one iterative process,* USSR Comp. Math. and Math. Phys. 4 (1964), 227–235.

[15] H. Gomez, T.J.R. Hughes, X. Nogueira, and V.M. Calo: *Isogeometric analysis of the isothermal Navier-Stokes-Korteweg equations,* Comput. Methods Appl. Mech. Engrg. 199 (2010), 1828–1840.

[16] W. Hackbusch: *Multi-Grid Methods and Applications,* Springer, 1985.

[17] Klaus Höllig, *Finite Element Methods with B-Splines,* SIAM, 2003.

[18] K. Höllig, J. Hörner, and A. Hoffacker: *Finite element analysis with b-splines: weighted and isogeometric methods,* to appear in Proc. Curves and Surfaces, Avignon, 2010.

[19] K. Höllig, J. Hörner, and M. Pfeil: *Parallel finite element methods with weighted linear b-splines,* High Performance Computing in Science and Engineering 07, editors: W.E. Nagel, D. Kröner, and M. Resch, Springer 2008, 667–676.

[20] K. Höllig, U. Reif, and J. Wipper: *Weighted extended B-spline approximation of Dirichlet problems,* SIAM J. Numer. Anal. 39 (2001), 442–462.

[21] K. Höllig, U. Reif, and J. Wipper: *Multigrid methods with web-splines,* Numer. Math. 91 (2002), 237–256.

[22] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs: *Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement,* Comput. Methods Appl. Mech. Engrg. 194 (2005), 4135–4195.

[23] E. Kaygisiz: *FE-Approximation und Mehrgitterverfahren mit eB-Splines angewandt auf das Neumann Problem,* Diplomarbeit (in German), Universität Stuttgart, 2010.

[24] A. Kniehl: *FE-Verfahren für die Plattengleichung mit WEB-Splines und Mehrgitterverfahren,* Diplomarbeit (in German), Universität Stuttgart, 2010.

[25] J.M. Lane and R.F. Riesenfeld: *A theoretical development for the computer generation and display of piecewise polynomial surfaces,* IEEE Trans. Pattern Anal. Mach. Intellig. 2 (1980), 35–45.

[26] B. Mößner and U. Reif: *Stability of tensor product b-splines on domains,* J. Approx. Theory 154 (2008), 1–19.

[27] J. Peters and U. Reif: *Subdivision Surfaces,* Springer, Series Geometry and Computing, 2008.

[28] M. Pfeil: *WEB-Approximation elliptischer Eigenwertprobleme,* Dissertation (in German), Universität Stuttgart, 2007.

[29] L.L. Schumaker: *Spline Functions: Basic Theory,* Wiley-Interscience, New York, 1980.

[30] G. Strang and G.J. Fix: *An Analysis of the Finite Element Method,* Prentice–Hall, Englewood Cliffs, NJ, 1973.

[31] J. Wipper: *Finite-Elemente-Approximation mit WEB-Splines,* Dissertation (in German), Universität Stuttgart, 2005.

[32] O.C. Zienkiewicz and R.I. Taylor: *Finite Element Method,* Vol. I–III, Butterworth & Heinemann, London, 2000.

Klaus Höllig
Pfaffenwaldring 57
70569 Stuttgart
Germany
E-Mail: `hoellig@mathematik.uni-stuttgart.de`
WWW: `http://www.imng.uni-stuttgart.de/LstNumGeoMod/Hoellig/`

Jörg Hörner
Pfaffenwaldring 57
70569 Stuttgart
Germany
E-Mail: `hoernejg@mathematik.uni-stuttgart.de`
WWW: `http://www.imng.uni-stuttgart.de/LstNumGeoMod/Hoerner/`

Erschienene Preprints ab Nummer 2007/001

Komplette Liste: `http://www.mathematik.uni-stuttgart.de/preprints`

2011/016 *Höllig, K.; Hörner, J.:* Programming Multigrid Methods with B-Splines

2011/015 *Ferrario, P.:* Nonparametric Local Averaging Estimation of the Local Variance Function

2011/014 *Müller, S.; Dippon, J.:* k-NN Kernel Estimate for Nonparametric Functional Regression in Time Series Analysis

2011/013 *Knarr, N.; Stroppel, M.:* Unitals over composition algebras

2011/012 *Knarr, N.; Stroppel, M.:* Baer involutions and polarities in Moufang planes of characteristic two

2011/011 *Knarr, N.; Stroppel, M.:* Polarities and planar collineations of Moufang planes

2011/010 *Jentsch, T.; Moroianu, A.; Semmelmann, U.:* Extrinsic hyperspheres in manifolds with special holonomy

2011/009 *Wirth, J.:* Asymptotic Behaviour of Solutions to Hyperbolic Partial Differential Equations

2011/008 *Stroppel, M.:* Orthogonal polar spaces and unitals

2011/007 *Nagl, M.:* Charakterisierung der Symmetrischen Gruppen durch ihre komplexe Gruppenalgebra

2011/006 *Solanes, G.; Teufel, E.:* Horo-tightness and total (absolute) curvatures in hyperbolic spaces

2011/005 *Ginoux, N.; Semmelmann, U.:* Imaginary Khlerian Killing spinors I

2011/004 *Scherer, C.W.; Köse, I.E.:* Control Synthesis using Dynamic D-Scales: Part II — Gain-Scheduled Control

2011/003 *Scherer, C.W.; Köse, I.E.:* Control Synthesis using Dynamic D-Scales: Part I — Robust Control

2011/002 *Alexandrov, B.; Semmelmann, U.:* Deformations of nearly parallel G_2-structures

2011/001 *Geisinger, L.; Weidl, T.:* Sharp spectral estimates in domains of infinite volume

2010/018 *Kimmerle, W.; Konovalov, A.:* On integral-like units of modular group rings

2010/017 *Gauduchon, P.; Moroianu, A.; Semmelmann, U.:* Almost complex structures on quaternion-Kähler manifolds and inner symmetric spaces

2010/016 *Moroianu, A.; Semmelmann,U.:* Clifford structures on Riemannian manifolds

2010/015 *Grafarend, E.W.; Kühnel, W.:* A minimal atlas for the rotation group $SO(3)$

2010/014 *Weidl, T.:* Semiclassical Spectral Bounds and Beyond

2010/013 *Stroppel, M.:* Early explicit examples of non-desarguesian plane geometries

2010/012 *Effenberger, F.:* Stacked polytopes and tight triangulations of manifolds

2010/011 *Györfi, L.; Walk, H.:* Empirical portfolio selection strategies with proportional transaction costs

2010/010 *Kohler, M.; Krzyżak, A.; Walk, H.:* Estimation of the essential supremum of a regression function

2010/009 *Geisinger, L.; Laptev, A.; Weidl, T.:* Geometrical Versions of improved Berezin-Li-Yau Inequalities

2010/008 *Poppitz, S.; Stroppel, M.:* Polarities of Schellhammer Planes

2010/007 *Grundhöfer, T.; Krinn, B.; Stroppel, M.:* Non-existence of isomorphisms between certain unitals

2010/006 *Höllig, K.; Hörner, J.; Hoffacker, A.:* Finite Element Analysis with B-Splines: Weighted and Isogeometric Methods