NFN - Nationales Forschungsnetzwerk

Geometry + Simulation

http://www.gs.jku.at

A Black-Box Algorithm for Fast Matrix Assembly in Isogeometric Analysis

Clemens Hofreither

G+S Report No. 55

April 2017

Der Wissenschaftsfonds.

JOHANNES KEPLER UNIVERSITY LINZ

A Black-Box Algorithm for Fast Matrix Assembly in Isogeometric Analysis

Clemens Hofreither*

April 18, 2017

Abstract

We present a fast algorithm for assembling stiffness matrices in Isogeometric Analysis with tensor product spline spaces. The procedure exploits the facts that (a) such matrices have block-banded structure, and (b) they often have low Kronecker rank. Combined, these two properties allow us to reorder the nonzero entries of the stiffness matrix into a relatively small, dense matrix or tensor of low rank. A suitable black-box low-rank approximation algorithm is then applied to this matrix or tensor. This allows us to approximate the nonzero entries of the stiffness matrix while explicitly computing only relatively few of them, leading to a fast assembly procedure.

The algorithm does not require any further knowledge of the used spline spaces, the geometry transform, or the partial differential equation, and thus is black-box in nature. Existing assembling routines can be reused with minor modifications. A reference implementation is provided which can be integrated into existing code.

Numerical examples demonstrate significant speedups over a standard Gauss quadrature assembler for several geometries in two and three dimensions. The runtime scales sublinearly with the number of degrees of freedom in a large pre-asymptotic regime.

1 Introduction

An often-cited obstacle in the practical use of Isogeometric Analysis (IgA; [10]) is the high computational effort required to assemble the involved stiffness matrices, especially for higher spline degrees. For a d-dimensional tensor product spline space with n^d degrees of freedom and spline degree p, standard assemblers based on tensor product Gauss quadrature require $\mathcal{O}(n^d p^{3d})$ operations to compute the stiffness matrix, which has $\mathcal{O}(n^d p^d)$ nonzero entries.

For this reason, various approaches have been proposed to speed up matrix asssembly in IgA. Rather than attempt to reiterate all prior work, we refer to the two recent articles [9, 14], whose introductory sections contain an overview of the state of the art. In short, many approaches aim at developing quadrature

*Department of Computational Mathematics, Johannes Kepler University Linz, Altenbergerstr. 69, 4040 Linz, Austria. <chofreither@numa.uni-linz.ac.at>

The author was supported by the National Research Network "Geometry + Simulation" (NFN S117), funded by the Austrian Science Fund (FWF).

rules which are more efficient than the naive tensor product Gauss quadrature approach with $\mathcal{O}(p)$ nodes per knot span. On the other hand, the method presented in [14] is based on techniques of low-rank tensor approximation. Although its applicability depends on the existence of good low-rank approximations to the geometry map and coefficient functions, these techniques appear to be the only known ones to achieve running times that scale sublinearly with the number of degrees of freedom in a significant pre-asymptotic regime. For surveys of the large field of low-rank tensor methods, see [12, 11, 6].

Much like the approach in [14], the algorithm presented in the present work is based on low-rank approximation and shares many of the same properties. In particular, it achieves comparable speedups to those claimed in [14], can achieve sublinear scaling, and its efficiency depends significantly on the low-rank approximability of the involved quantities. The major advantage of the proposed method is that, in contrast to [14], is is purely algebraic and black-box in nature:

- it does not require explicit knowledge of the PDE, the geometry map or the used basis functions and works without modifications for a wide class of problems;
- it can reuse existing assembler routines with minor modifications;
- it is much easier to implement;
- a black-box implementation can be provided which can be integrated into existing codebases relatively easily.

Rather than an entirely new assembler, the proposed algorithm may thus be considered an algebraic "accelerator" to be wrapped around existing assemblers. Two open-source implementations of this algorithm are made available to the community (see Section 5.1).

Our approach is based on a matrix reordering first described by Van Loan and Pitsianis [17] which converts Kronecker products into outer products of vectors, and matrices with low Kronecker rank into matrices with low rank. The particular block structure of IgA matrices allows us to eliminate the sparsity pattern of the reordered matrix, producing a relatively small, dense matrix which contains exactly the nonzeros of the original matrix. Since IgA matrices often have low Kronecker rank or can be well approximated by matrices with low Kronecker rank, the reordered matrix typically has low rank. By applying Adaptive Cross Approximation (ACA) [1, 2] to this low-rank matrix, we can compute it very accurately while evaluating only relatively few of its entries. Once this is done, we recover the original stiffness matrix by simply reverting the reordering operation.

The idea of the matrix reordering from [17] was already exploited for approximations with low Kronecker rank in [8]; there, however, the Kronecker factors were not banded, but instead approximated using $\mathcal{H}$-matrices (cf. [7, 2]). A similar approach where the blocks have Toeplitz structure is given in [15]. Tyrtyshnikov [16] gives some error estimates for a related approximation problem. The exploitation of the block-banded structure that we use to represent the reordered matrix compactly appears to be novel; likewise the application of ACA to the reordered matrix in order to recover Kronecker approximations. A

previous use of ACA in the IgA context was for the efficient approximation of bivariate functions by sums of separable splines [4].

The remainder of the paper is structured as follows. In Section 2, we describe two crucial but natural properties of IgA stiffness matrices which we exploit in the construction of our algorithm, namely the particular block-banded structure and the low numerical Kronecker rank. Based on these two properties, we develop our fast assembling algorithm in the 2D case in Section 3. The extension to the 3D case is then rather straightforward and is described in Section 4. We discuss an implementation and present numerical examples in Section 5.

2 Properties of IgA stiffness matrices

2.1 Hierarchical block-banded structure

We choose first a set of univariate basis functions

$$\Phi = (\varphi_i)_{i=1}^n$$

over the interval $[0,1]$. We make the assumption that the basis functions have local support in the sense that, for some $p \in \mathbb{N}$, we have

$$|i-j| > p \quad \Longrightarrow \quad |\operatorname{supp}\varphi_i \cap \operatorname{supp}\varphi_j| = 0 \qquad \forall i,j = 1,\dots,n.$$

Basis functions typically used in IgA, such as B-splines and NURBS, naturally satisfy this assumption with p being the spline degree. In higher dimensions, we take tensor product bases, e.g., with univariate bases Φ_1 and Φ_2,

$$\varphi_{i_1,i_2} \in \Phi_1 \times \Phi_2 : \quad \varphi_{i_1,i_2}(x,y) = \varphi_{i_1}(x)\varphi_{i_2}(y) \tag{1}$$

in the two-dimensional setting. If Φ_1 and Φ_2 have support overlap p_1 and p_2, respectively, then the tensor product basis inherits a similar local support property, namely,

$$|i_1 - j_1| > p_1 \vee |i_2 - j_2| > p_2 \quad \Longrightarrow \quad |\operatorname{supp}\varphi_{i_1,i_2} \cap \operatorname{supp}\varphi_{j_1,j_2}| = 0 \tag{2}$$

Typically in IgA, one introduces a geometry mapping $\mathcal{G} : [0,1]^d \to \Omega$ from the parameter domain to the computational domain of interest. The actual basis functions over Ω are then given by

$$\widetilde{\Phi} = \{\varphi \circ \mathcal{G}^{-1} : \varphi \in \Phi\}.$$

The basis functions so defined may lose the tensor product property (1), but retain the local support property (2).

We now assume to be given a bilinear form $a(\cdot,\cdot)$, defined at least over the span of the basis functions, which preserves locality in the sense that

$$|\operatorname{supp} u \cap \operatorname{supp} v| = 0 \quad \Longrightarrow \quad a(u,v) = 0. \tag{3}$$

Typically, $a(\cdot,\cdot)$ arises from the weak formulation of a partial differential equation of interest, in which case this condition is natural.

Let $N = |\Phi|$ denote the total number of basis functions. We are interested in the fast computation of the matrix

$$A = (a_{ij})_{i,j=1}^N, \qquad a_{ij} = a(\varphi_j, \varphi_j)$$

which represents the discretization of the bilinear form $a(\cdot,\cdot)$. Basis functions arising from a tensor product construction are assumed to be numbered in lexicographic order. We will always refer to A as a *stiffness matrix* in the sequel, although it may also be a mass matrix or arise from a different bilinear form.

Under the assumptions (3) and local support of the basis functions, A has

- banded structure in the 1D case,
- block-banded structure in the 2D case,
- a "multi-level" or "hierarchical" banded structure in the 3D and higher-dimensional case.

See Figure 1 for examples of 2D and 3D sparsity patterns. To make precise the structure of these matrices, we give the following inductive definitions.

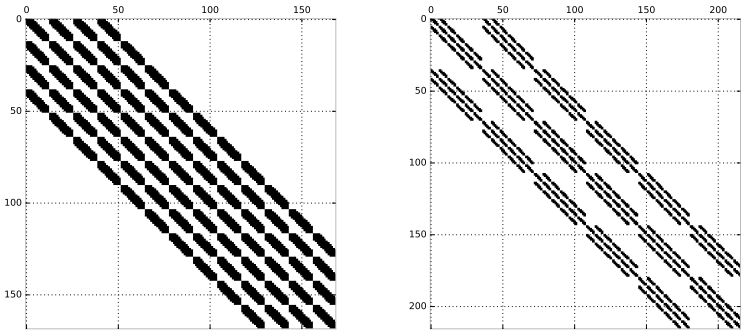


Figure 1: Sparsity pattern of a 2D (left) and a 3D (right) IgA stiffness matrix. The 2D matrix is a banded block matrix where each block is itself banded. The 3D matrix is a banded block matrix where each block has the structure of a 2D matrix. This suggests the inductive Definition 2.

Definition 1. A matrix $B \in \mathbb{R}^{n\times n}$ is called *banded* or *1-level banded* with bandwidth p if

$$|i-j| > p \implies b_{ij} = 0 \qquad \forall i,j = 1,\dots,n.$$

Definition 2. A matrix $A \in \mathbb{R}^{n_1\cdots n_d \times n_1 \cdots n_d}$ with blocks

$$A_{ij} \in \mathbb{R}^{n_2\cdots n_d \times n_2\cdots n_d}, \qquad i,j \in \{1,\dots,n_1\}$$

is called *d-level banded* with bandwidths $(p_1,\dots,p_d)$ if each block A_{ij} is $(d-1)$-level banded with bandwidths $(p_2,\dots,p_d)$ and

$$|i-j| > p_1 \implies A_{ij} = 0 \qquad \forall i,j \in \{1,\dots,n_1\}.$$

It is easy to see that, by the above assumptions, d-dimensional IgA stiffness matrices are d-level banded due to the local support property.

2.2 Kronecker product representations of IgA matrices

Many classes of matrices arising in IgA have natural representations in terms of Kronecker products of lower-dimensional matrices. For instance, for any d-dimensional tensor product basis

$$\Phi = \Phi_1 \times \ldots \times \Phi_d$$

over $[0,1]^d$, the mass matrix can be written as the Kronecker product

$$M^{(d)} = M_1 \otimes \ldots \otimes M_d$$

of the univariate mass matrices M_j for Φ_j. The stiffness matrix for the Poisson equation in $[0,1]^d$ can be written inductively as

$$\begin{aligned} K^{(1)} &= K_1, \\ K^{(d+1)} &= M^{(d)} \otimes K_{d+1} + K^{(d)} \otimes M_{d+1}, \end{aligned}$$

where M_j and K_j are the univariate mass and stiffness matrices, respectively, for Φ_j. Expanding this recurrence, we find that $K^{(d)}$ can be written as a sum of d Kronecker products with d factors each. No representation with fewer Kronecker products of these sizes exists. We say that $M^{(d)}$ has *Kronecker rank* 1, and $K^{(d)}$ has *Kronecker rank* d.

Once geometry transforms or varying PDE coefficients enter the picture, these simple relations do not hold anymore. Nevertheless, if geometry transform and coefficients are sufficiently smooth, good approximations to the mass and stiffness matrices as sums of relatively few Kronecker products typically exist.

If k is the smallest integer such that A can be approximated to a fixed accuracy ε by a sum of k Kronecker products, we say that A has *numerical Kronecker rank* k. This is analogous to the standard notions of rank and numerical rank (cf. [5]). There are many large matrices (such as those obtained by sampling smooth functions on tensor product grids) with high or full rank, but low numerical rank. These matrices admit good low-rank approximations. Similarly, many block-structured IgA matrices admit good approximations with low Kronecker rank.

The numerical Kronecker ranks for 3D IgA stiffness matrices over different geometries were estimated numerically in [14]. In the tested examples, the numerical Kronecker rank never exceeded 37. Crucially, it remained uniformly bounded with respect to mesh refinements and increases in the spline degree.

A rigorous analytical bound on the numerical Kronecker rank of IgA stiffness matrices is not known yet and is an interesting topic for future research.

3 Fast assembling in 2D

3.1 A reordering operation for block matrices

We denote by

$$\mathrm{vec} : \mathbb{R}^{m \times n} \to \mathbb{R}^{mn}$$

the row-wise vectorization operator which maps a matrix to a vector by concatenating all its rows.

Remark 1. In [17], the column-wise vectorization operator was used. However, the particular choice of the order does not matter, and we use the row-wise order in order to better correspond to the lexicographic order in the 3D case that we treat later on.

For a matrix $A \in \mathbb{R}^{mn \times mn}$ with blocks

$$A_{ij} \in \mathbb{R}^{n \times n}, \qquad i, j \in \{1, \dots, m\},$$

we define the reordering

$$\mathcal{R}(A) = \mathcal{R}_{m,n}(A) = \begin{pmatrix} (\operatorname{vec} A_{11})^T \\ (\operatorname{vec} A_{12})^T \\ \vdots \\ (\operatorname{vec} A_{m,m-1})^T \\ (\operatorname{vec} A_{m,m})^T \end{pmatrix} \in \mathbb{R}^{m^2 \times n^2}$$

as the matrix whose rows are the vectorizations of the blocks of A, ordered lexicographically. Although $\mathcal{R} = \mathcal{R}_{mn}$ depends on the block sizes m and n, these sizes will always be clear from context and we omit them in writing.

This reordering operation was first described by Van Loan and Pitsianis [17]. The following fundamental property of the reordering of a Kronecker product, although never explicitly stated in [17], is the motivation for its definition.

Theorem 1. *For arbitrary matrices $B \in \mathbb{R}^{m \times m}, C \in \mathbb{R}^{n \times n}$, we have*

$$\mathcal{R}(B \otimes C) = (\operatorname{vec} B)(\operatorname{vec} C)^T.$$

Proof. Follows immediately from the fact that the blocks of the Kronecker product $A = B \otimes C$ are given by $A_{ij} = b_{ij}C$, and hence $\operatorname{vec}(A_{ij}) = b_{ij} \operatorname{vec} C$. □

The above result states that a Kronecker product, when reordered, turns into a rank 1 matrix. More generally, due to linearity, if A can be written as a sum of K Kronecker products $B_i \otimes C_i$, then its reordering is given by

$$\mathcal{R}(A) = \mathcal{R}\left(\sum_{i=1}^{K} B_i \otimes C_i\right) = \sum_{i=1}^{K} (\operatorname{vec} B_i)(\operatorname{vec} C_i)^T. \tag{4}$$

The smallest integer K for which we can find a representation of $\mathcal{R}(A)$ of the form (4) is the rank of $\mathcal{R}(A)$. Equivalently, it is the *Kronecker rank* of A, i.e., the smallest K such that A is a sum of K Kronecker products (of matrices with sizes m and n). In this sense, $\mathcal{R}$ converts Kronecker rank into matrix rank.

We observe that the Frobenius norm (and any other matrix norm defined entrywise) is invariant under $\mathcal{R}$. Therefore, $\mathcal{R}$ allows us to convert good Kronecker product approximations into good low-rank approximations and vice versa.

3.2 Reordering the stiffness matrix

We now apply the reordering operator from Section 3.1 to stiffness matrices arising from 2D IgA discretizations as described in Section 2.1. We assume that the used basis is the tensor product of univariate bases with dimensions n_1, n_2 and support overlaps p_1, p_2, respectively. Recall that the resulting matrix

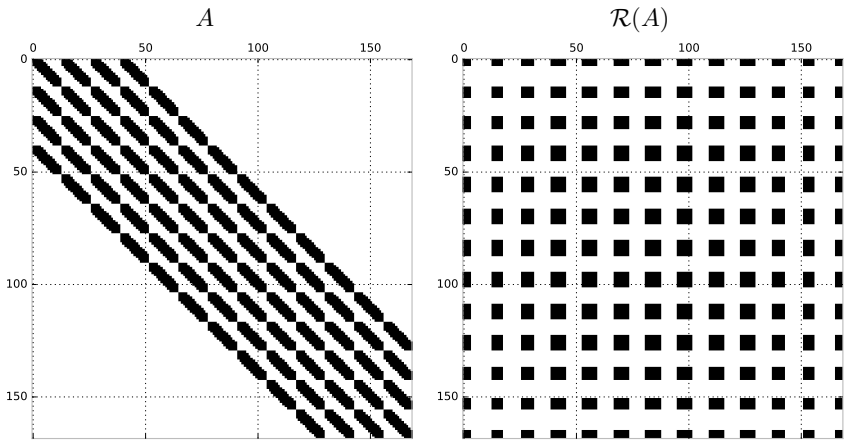


Figure 2: Sparsity structure of a 2D IgA stiffness matrix (left) and of its reordering (right).

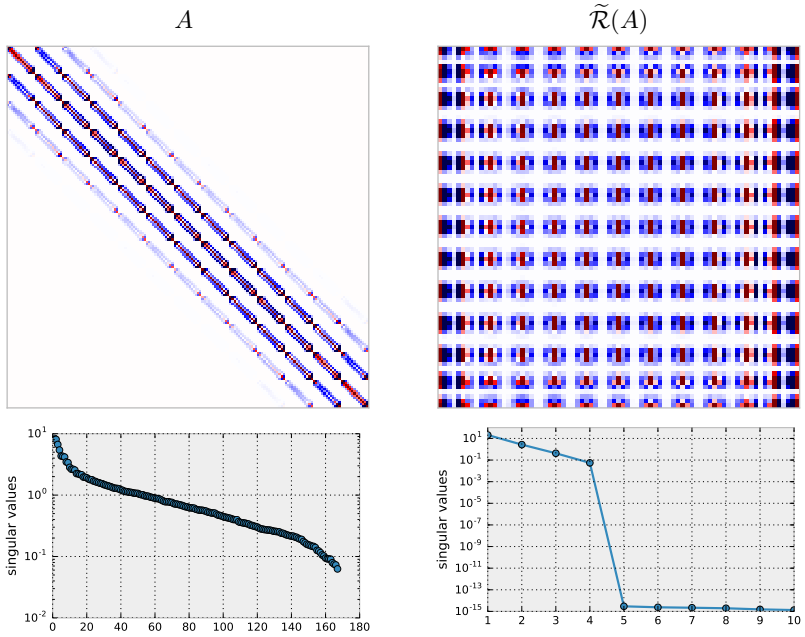


Figure 3: Entries (top) and singular values (bottom) of a 2D IgA stiffness matrix (left) and its deflated reordering (right). The geometry map is the quarter annulus from Example 2D-A (see Section 5.2).

A is 2-level banded in the sense of Definition 2 with bandwidths (p_1, p_2). The sparsity patterns of A and of its reordering $\mathcal{R}(A)$ are illustrated in Figure 2. We observe that the resulting structure is very particular. This can be explained as follows:

- Each row of $\mathcal{R}(A)$ is the vectorization of one block A_{ij} of A. Due to Definition 2, we have $|i-j| > p_1 \implies A_{ij} = 0$. Therefore, only $\mu_1 = \mathcal{O}(n_1 p_1)$ rows of $\mathcal{R}(A)$ are nonzero. We can precompute this pattern since it only depends on n_1 and p_1.
- Due to Definition 2, each block $A_{ij} \in \mathbb{R}^{n \times n}$ has bandwidth p_2. Thus, each nonzero row of $\mathcal{R}(A)$ contains $\mu_2 = \mathcal{O}(n_2 p_2)$ nonzero entries. Again, we can precompute the pattern in dependence of n_2 and p_2.

This shows that we can easily extract a $\mu_1 \times \mu_2$ submatrix of $\mathcal{R}(A)$ which contains all its nonzero entries. We denote this dense, "deflated" submatrix by

$$\widetilde{\mathcal{R}}(A) \in \mathbb{R}^{\mu_1 \times \mu_2}.$$

Note that its entries are merely the $\mu_1 \mu_2 = \mathcal{O}(n^2 p^2)$ nonzero entries of A in a different order.

We assume that the stiffness matrix A has low (numerical) Kronecker rank due to the considerations in Section 2.2. It follows from Theorem 1 that $\mathcal{R}(A)$, and hence also $\widetilde{\mathcal{R}}(A)$, have low (numerical) rank. As an example, see Figure 3, where A and $\widetilde{\mathcal{R}}(A)$ are plotted together with their singular values. Here, A is the stiffness matrix for a quarter annulus domain. In this example, $\widetilde{\mathcal{R}}(A)$ has rank 4.

Dense matrices with low rank, such as $\widetilde{\mathcal{R}}(A)$, can be well approximated while computing only relatively few of their entries by black-box algorithms such as ACA, which we describe in the next section.

3.3 The ACA algorithm

The ACA algorithm was first described by Bebendorf [1, 2] and has been widely used for the data-sparse approximation of dense matrices arising in 3D boundary element methods. We give a basic version of it in Algorithm 1. Its core idea is to construct a low-rank approximation by a series of greedily chosen rank 1 updates.

Algorithm 1 Adaptive Cross Approximation (ACA)

Given: $B \in \mathbb{R}^{m \times n}$
Output: $X \in \mathbb{R}^{m \times n}$

$X \leftarrow 0$, $E \leftarrow B$
for $k = 1, \ldots, K$ **do**
  choose a pivot (i, j) with $e = E[i, j] \neq 0$
  compute the row $r = E[i, :]$
  compute the column $c = E[:, j]$
  update error $E \leftarrow E - \frac{1}{e} c r^T$
  update approximation $X \leftarrow X + \frac{1}{e} c r^T$
end for

Here and in the following, we use the notation $E[i, j]$ to refer to individual entries of a matrix, and $E[i, :]$ and $E[:, j]$ for the i-th row and the j-th column, respectively.

Some remarks on the implementation of ACA are in order.

- The input matrix B is typically not given in array form, but as a routine which can compute individual entries of B on demand.
- In practice, one does not store the error matrix E, but instead computes it on the fly from the invariant $E = B - X$. Therefore, accessing one entry of E incurs the computation of one entry of B.
- Likewise, in many implementations, X is not stored as a full matrix, but instead as a list of already computed rank 1 approximations. This allows the reduction of the complexity from $\mathcal{O}(Kmn)$ to $\mathcal{O}(K^2(m+n))$. However, since we finally require all entries of X anyway, we do not use this approach here and instead store X as a full matrix.
- The strategy for choosing the pivot $E[i, j]$ is crucial to good performance. For a good convergence rate, it is desirable to choose pivots with large absolute values (see the analysis given in [2]). For instance, a *full pivoting* strategy searches for the entry of the error matrix E with largest absolute value. This means, however, that all entries of B have to be generated. More economical is a *row pivoting* strategy, where the maximum is only sought over a single row of E. After computing the column c, a heuristic for the next row to be chosen is to take the row where c has maximum absolute value.

  While heuristic in nature, row pivoting is known to perform very well in many applications, and we use it in our implementation. See [4] for a favorable numerical comparison of the convergence rates of truncated SVD, ACA with full pivoting, and ACA with row pivoting.
- The stopping criterion is given as a fixed number of iterations K in Algorithm 1. In practice, one can specify a desired tolerance ε and terminate the algorithm adaptively. Our strategy is the following:
  - If $|e|$ is close to machine epsilon, skip the current row for stability reasons and choose a new one by random. If this happens three times in a row, terminate.
  - Otherwise, if $|e| < \varepsilon$ for three iterations in a row, terminate.

With an implementation as described above, ACA requires the computation of $\mathcal{O}(K(m+n))$ entries of B. The overall complexity is $\mathcal{O}(K(m+n)\gamma + Kmn)$, where γ is the cost for computing one entry of the input matrix.

3.4 The fast assembling algorithm in 2D

We now have all components in place to state the fast assembling algorithm in the 2D case. The simple procedure is given in Algorithm 2.

In the first step, $\widetilde{\mathcal{R}}(A)$ is approximated by means of ACA (Algorithm 1). Recall that ACA does not require the full input matrix, but only a routine which computes individual entries on demand. A procedure to compute an entry $\widetilde{\mathcal{R}}(A)[i, j]$ needs two ingredients:

Algorithm 2 Fast assembling algorithm in 2D

1. Compute $\widetilde{X} = \texttt{ACA}(\widetilde{\mathcal{R}}(A)) \in \mathbb{R}^{\mu_1 \times \mu_2}$
2. Reconstruct $\widetilde{A} = \widetilde{\mathcal{R}}^{-1}(\widetilde{X}) \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$

- A translation of the indices $(i, j) \mapsto (i^*, j^*)$ such that $\widetilde{\mathcal{R}}(A)[i, j] = A[i^*, j^*]$. This is easily achieved by a simple re-indexing calculation. See (7) in Section 4.1 for details.
- A routine to evaluate the stiffness matrix entry $A[i^*, j^*] = a(\varphi_{j^*}, \varphi_{i^*})$. This is the only step which explicitly depends on the PDE, the geometry transform and the used basis functions. An existing assembler routine (e.g., based on tensor product Gauss quadrature) can typically be easily modified to produce a single entry instead of the entire matrix A.

Once $\widetilde{X}$ has been computed to the desired accuracy by ACA, in the second step the reordering and deflation operation $\widetilde{\mathcal{R}}$ is reversed to produce an approximation $\widetilde{A}$ to the original stiffness matrix A. Again, this is easily done by basic index calculations.

Error analysis. In the Frobenius norm (or any other matrix norm defined entrywise), we have

$$\|A - \widetilde{A}\|_F = \|\widetilde{\mathcal{R}}(A) - \widetilde{\mathcal{R}}(\widetilde{A})\|_F = \|\widetilde{\mathcal{R}}(A) - \widetilde{X}\|_F.$$

The latter term is just the error of the ACA approximation. It can be controlled (heuristically) by choosing the target accuracy ε in the ACA algorithm. Our experiments in Section 5.2 confirm that usually $\|A - \widetilde{A}\|_F \lesssim \varepsilon$ and $\|A - \widetilde{A}\|_2 \lesssim \varepsilon$.

Complexity analysis. The complexity is dominated by the ACA step. Assuming $n_1 = n_2 = n$ and $p_1 = p_2 = p$, the size of $\widetilde{\mathcal{R}}(A)$ is $\mu \times \mu$ with $\mu = \mathcal{O}(np)$ (cf. Section 3.2). Hence, by the ACA complexity analysis in Section 3.3, we obtain the complexity $\mathcal{O}(Knp\gamma + Kn^2p^2)$, where γ is the cost for computing one entry of the stiffness matrix. Using a typical tensor product Gauss quadrature rule with $\mathcal{O}(p)$ quadrature nodes per knot span, we have $\gamma = \mathcal{O}(p^4)$. This puts the overall complexity for Algorithm 2 at

$$\mathcal{O}(K(np^5 + n^2p^2)),$$

whereas standard Gauss quadrature assemblers behave like $\mathcal{O}(n^2p^6)$. In practice, the ACA rank K will depend on the desired accuracy ε as well as the (numerical) Kronecker rank of A. No analytical bounds for the latter are known, but we see some typical values for K in the examples in Section 5 as well as in [14]. Crucially, K is usually uniformly bounded with respect to n and p.

4 Fast assembling in 3D

The algorithm from Section 3 is rather straightforward to generalize to the 3D and higher-dimensional cases. We only require a suitable generalization of the reordering operator, which we provide in Section 4.1, and a generalization of ACA to 3D, which we describe in Section 4.2. The complete algorithm is then described in Section 4.3.

4.1 Matrix reordering in 3D

We first introduce an index-based notation for the previous reordering operation, which will make it obvious how to generalize it to higher dimensions.

For any $m, n \in \mathbb{N}$, denote the sequential numbering of the pairs (i, j) in lexicographical order by

$$(i,j)_{m,n} := (i-1)n + j \qquad \forall i = 1,\dots,m,\ j = 1,\dots,n. \tag{5}$$

For a block matrix $A \in \mathbb{R}^{mn\times mn}$, this notation allows us to describe by

$$A[(i_1,j_1)_{m,n},(i_2,j_2)_{m,n}], \qquad i_1,i_2 \in \{1,\dots,m\},\ j_1,j_2 \in \{1,\dots,n\},$$

the entry (j_1, j_2) of the block A_{i_1,i_2}. In particular, for two square matrices $B \in \mathbb{R}^{m\times m}, C \in \mathbb{R}^{n\times n}$, the Kronecker product $B \otimes C \in \mathbb{R}^{mn\times mn}$ satisfies

$$(B\otimes C)[(i_1,j_1)_{m,n},(i_2,j_2)_{m,n}] = B[i_1,i_2]C[j_1,j_2]. \tag{6}$$

The matrix reordering introduced in Section 3.1 can now be written as

$$\mathcal{R}(A)[(i_1,i_2)_{m,m},(j_1,j_2)_{n,n}] = A[(i_1,j_1)_{m,n},(i_2,j_2)_{m,n}]. \tag{7}$$

We point out that from (7) and (6), it immediately follows

$$\mathcal{R}(B\otimes C)[(i_1,i_2)_{m,m},(j_1,j_2)_{n,n}] = B[i_1,i_2]C[j_1,j_2],$$

which is merely a rewriting of the statement of Theorem 1.

Relation (7) allows a simple generalization of the reordering $\mathcal{R}$ to dimensions higher than two. The result will now no longer be a matrix, but a tensor of higher order. We first point out that the lexicographic numbering scheme introduced in (5) generalizes straightforwardly to higher dimensions and write

$$\begin{aligned}(i_1,i_2,i_3)_{n_1,n_2,n_3} := [(i_1-1)n_2 + (i_2-1)]\,n_3 + i_3,\\ i_1 = 1,\dots,n_1,\ i_2 = 1,\dots,n_2,\ i_3 = 1,\dots,n_3\end{aligned}$$

for such an index in three dimensions.

Definition 3. For a matrix $A \in \mathbb{R}^{n_1n_2n_3\times n_1n_2n_3}$, we define the reordering to a three-way tensor

$$\mathcal{R}(A) = \mathcal{R}_{n_1,n_2,n_3}(A) \in \mathbb{R}^{n_1^2\times n_2^2\times n_3^2}$$

by the relation

$$\begin{aligned}&\mathcal{R}(A)[(i_1,j_1)_{n_1,n_1},(i_2,j_2)_{n_2,n_2},(i_3,j_3)_{n_3,n_3}]\\ &= A[(i_1,i_2,i_3)_{n_1,n_2,n_3},(j_1,j_2,j_3)_{n_1,n_2,n_3}]\\ &\quad \forall\ i_1,j_1 = 1,\dots,n_1,\ i_2,j_2 = 1,\dots,n_2,\ i_3,j_3 = 1,\dots,n_3.\end{aligned}$$

For square matrices $B \in \mathbb{R}^{n_1\times n_1}, C \in \mathbb{R}^{n_2\times n_2}, D \in \mathbb{R}^{n_3\times n_3}$, we have, analogously to the 2D case, that their Kronecker product $A = B \otimes C \otimes D$ satisfies

$$\mathcal{R}(A)[(i_1,j_1)_{n_1,n_1},(i_2,j_2)_{n_2,n_2},(i_3,j_3)_{n_3,n_3}] = B[i_1,j_1]C[i_2,j_2]D[i_3,j_3].$$

That is, $\mathcal{R}(B\otimes C\otimes D)$ can be written as the outer product of three vectors, namely, of the vectorizations of B, C, and D. Thus, as in the 2D case, stiffness matrices of low Kronecker rank give rise to reordered tensors of low rank.

Let now $A \in \mathbb{R}^{n_1 n_2 n_3 \times n_1 n_2 n_3}$ be a 3-level banded matrix with bandwidths (p_1, p_2, p_3) according to Definition 2. We saw that typical IgA stiffness matrices have this property. Take arbitrary multi-indices

$$(i_1, i_2, i_3), (j_1, j_2, j_3) \in \{1, \ldots, n_1\} \times \{1, \ldots, n_2\} \times \{1, \ldots, n_3\}.$$

Due to Definition 2 and Definition 3, we have

$$(\exists \alpha \in \{1,2,3\} : |i_\alpha - j_\alpha| > p_\alpha) \quad \Longrightarrow \\ \mathcal{R}(A)[(i_1, j_1)_{n_1,n_1}, (i_2, j_2)_{n_2,n_2}, (i_3, j_3)_{n_3,n_3}] = 0.$$

Therefore, as in the 2D case, the reordered tensor $\mathcal{R}(A)$ has a rank 1 sparsity pattern with $\mu_j = \mathcal{O}(n_j p_j)$ nonzeros per direction. We can precompute this sparsity pattern and, after eliminating it from $\mathcal{R}(A)$, obtain a dense tensor

$$\widetilde{\mathcal{R}}(A) \in \mathbb{R}^{\mu_1 \times \mu_2 \times \mu_3}.$$

4.2 Cross approximation for three-way tensors

To perform the low-rank approximation of the three-way tensor which results from reordering the stiffness matrix, we require an extension of the ACA algorithm to tensors of higher order. Several such extensions have been published; we use the one by Bebendorf [3]. We give the basic form in Algorithm 3.

Algorithm 3 ACA3D [3]

Given: $B \in \mathbb{R}^{n_1 \times n_2 \times n_3}$
Output: $X \in \mathbb{R}^{n_1 \times n_2 \times n_3}$

$X \leftarrow 0$, $E \leftarrow B$
for $k = 1, \ldots, K$ **do**
  choose a pivot (i, j, k) with $e = E[i, j, k] \neq 0$
  compute the column $c = E[:, j, k]$
  approximate the matrix $V = \texttt{ACA}(E[i, :, :])$
  update error $E \leftarrow E - \frac{1}{e}(c \times V)$
  update approximation $X \leftarrow X + \frac{1}{e}(c \times V)$
end for

Here, the idea is to perform successive updates not with the outer product of vectors, as in the 2D case, but with the tensor product of a vector and a matrix, by which we mean

$$(c \times V)[i, j, k] = c[i]V[j, k].$$

Here, $V \in \mathbb{R}^{n_2 \times n_3}$ is the matrix slice obtained by fixing the first index of $E \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ to i. However, computing the entire matrix slice V would incur the computation of $n_2 n_3$ entries of the input tensor. In order to avoid this, V is not computed exactly, but by an application of standard matrix ACA (Algorithm 1). The ACA3D algorithm can thus be viewed as performing a hierarchical low-rank approximation.

Most of the remarks on implementation details from Section 3.3 remain valid for the 3D case. In particular, the pivoting strategy and stopping criteria

can be straightforwardly adapted to the 3D case. We found that using the same truncation rank K (or the same tolerance ε for adaptive stopping) for the inner ACA iteration as for the outer ACA3D iteration works well. In this case, we need to compute $\mathcal{O}(K(n_1 + K(n_2 + n_3)))$ entries of the input tensor. For $n_1 = n_2 = n_3 = n$, we obtain the bound $\mathcal{O}(K^2 n)$ for the number of computed entries and thus the overall complexity $\mathcal{O}(K^2 n\gamma + Kn^3)$ for ACA3D, where γ is the cost for computing one entry of the input tensor B.

We mention one possible improvement which seems not to have been described in the literature thus far: when invoking the inner ACA iteration to approximate the matrix slice $B[i,:,:]$, its starting matrix can be chosen as the current approximation $X[i,:,:]$ instead of 0 as in Algorithm 1. This does not change the overall complexity, but reduces the required number of inner iterations towards the end of the ACA3D algorithm, when $X[i,:,:]$ is already a good approximation to $B[i,:,:]$, and can lead to noticeable speedups.

4.3 The fast assembling algorithm in 3D

The fast assembling algorithm in 3D proceeds exactly as the 2D variant in Algorithm 2, but replacing $\widetilde{\mathcal{R}}$ by the corresponding 3D reordering operator from Section 4.1 and replacing ACA by ACA3D from Section 4.2.

Complexity analysis. The complexity is dominated by the ACA3D step. Assuming $n_1 = n_2 = n_3 = n$ and $p_1 = p_2 = p_3 = p$, the size of $\widetilde{\mathcal{R}}(A)$ is $\mu \times \mu \times \mu$ with $\mu = \mathcal{O}(np)$. Hence, ACA3D requires $\mathcal{O}(K^2 np\gamma + Kn^3 p^3)$ operations, where γ is the cost for computing one entry of the stiffness matrix. Using tensor product Gauss quadrature with $\mathcal{O}(p)$ quadrature nodes per knot span, we have $\gamma = \mathcal{O}(p^6)$. This puts the overall complexity for the fast 3D assembling algorithm at

$$\mathcal{O}(K^2 np^7 + Kn^3 p^3),$$

whereas standard Gauss quadrature assemblers behave like $\mathcal{O}(n^3 p^9)$. We see some typical values for the rank K in the examples in Section 5.

5 Numerical examples

5.1 Implementation

Two implementations of the algorithm described in this paper were made by the author and are provided as open source software. Both can be found on the author's homepage[1].

The first is a standalone C++ file `fastasm.cc` without dependencies on any external libraries. It does not include routines for computing entries of IgA stiffness matrices, but is intended as a drop-in implementation in existing codebases where assembling routines already exist and can be modified to produce the stiffness matrix entrywise.

The second implementation is part of the Python research toolbox for Isogometric Analysis, `pyiga`[2]. It contains both a pure Python implementation of the algorithm as well as a wrapper around the C++ implementation `fastasm.cc`. The latter was used to produce the numerical examples below.

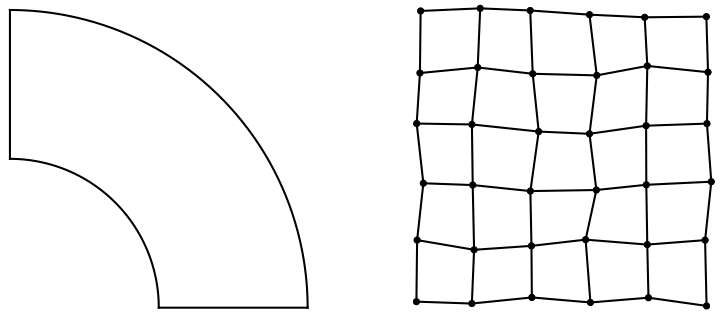
[1] `http://www.numa.uni-linz.ac.at/~chofreither/software/`
[2] `https://github.com/c-f-h/pyiga/`

The computation of the individual stiffness matrix entries in `pyiga` is implemented in Cython[3], a dialect of Python which compiles to C/C++. When used as a standard "full" assembler without acceleration, it was confirmed that these routines are relatively performant (slowdown no more than a factor 2x compared to the assemblers contained in the G+Smo library[4]). These are the numbers reported below for the full tensor product Gauss quadrature assembler.

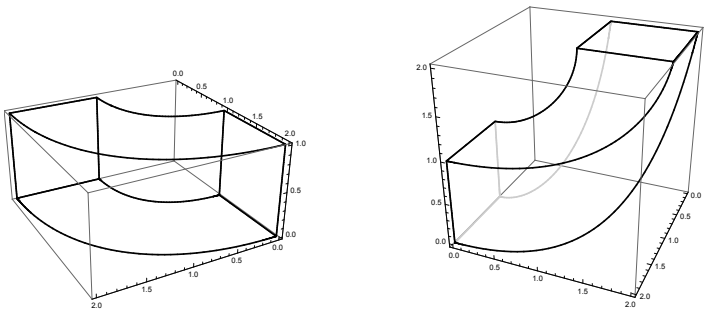
In addition, `pyiga` provides full multithreaded parallelization for all assembler routines. However, this feature was not used in the tests for clarity.

5.2 Tests

We compute Poisson stiffness matrices A for IgA with tensor product B-spline basis functions for the four computational domains shown in Figure 4. Each one is a single tensor product B-spline patch.



Example 2D-A Example 2D-B



Example 3D-A Example 3D-B

Figure 4: Example geometries

For each space dimension, one simpler and one more challenging domain was chosen. In 2D, the measure of complexity is simply the rank of $\mathcal{R}(A)$ or equivalently, the Kronecker rank of A. In 3D, since the approximation produced by ACA3D is closer in structure to a Tucker decomposition than to a canonical decomposition, we use the Tucker rank of $\mathcal{R}(A)$, estimated by the rank of the HOSVD [13], as a gauge of complexity. The number of outer and inner ACA3D

[3] `http://cython.org/`

[4] `http://www.gs.jku.at/gismo`

iterations never exceeded this number in our experiments. See [12] for a survey of these concepts.

- **Example 2D-A:** a quarter annulus-shaped domain, approximated with tensor product B-splines. The rank of $\mathcal{R}(A)$ is 4.
- **Example 2D-B:** a piecewise linear 6×6 discretization of the unit square whose control points are randomly perturbed by uniformly distributed values in $[-0.02, 0.02]^2$. Due to lack of smoothness, this domain is more challenging for low-rank approximation methods. The numerical rank of $\mathcal{R}(A)$ is uniformly bounded by 75.
- **Example 3D-A:** a cylindrical extension of the quarter annulus from Example 2D-A. The Tucker rank of $\mathcal{R}(A)$ is uniformly bounded by 5.
- **Example 3D-B:** a "twisted box" geometry which can be viewed as similar to Example 3D-A, but with one square face twisted upwards. Somewhat surprisingly, this example is significantly more challenging. The numerical Tucker rank of $\mathcal{R}(A)$ is uniformly bounded by 40.

We remark that in all cases, the mass matrices have significantly lower rank and are much easier and faster to compute than the stiffness matrices. We therefore omit them in our tests.

We compute the stiffness matrices for these domains using tensor product B-spline spaces with n uniform knot spans per direction and spline degree p. We report the timings for a standard tensor product Gauss quadrature assembler as well as for our accelerated low-rank assembler, where the desired accuracy for the ACA or ACA3D iteration was set to $\varepsilon = 10^{-10}$. We confirmed that in all cases, the final error in the spectral norm between the matrix produced by the standard assembler and the fast assembler was on the order of ε or less. Thus, a standard perturbation analysis shows that the error in the solution of a linear system with the approximated stiffness matrix relatively to the solution with the exact stiffness matrix can be made arbitrarily small by a proper choice of ε.

The runtimes obtained on a Linux workstation with an Intel® Core™ i7-2600 CPU with 3.40GHz and 8GB RAM, using only a single core, are shown in Table 1 and Table 2 for the 2D and 3D problems, respectively. For each test case, we give first the times for the Gauss quadrature assembler, then those for our fast low-rank assembler, and finally the speedup, i.e., the ratio of the times.

We observe that the speedups increase both with n and with p. Even for low spline degrees, however, the speedups are significant. They are particularly dramatic in Example 3D-A, a 3D problem with very low rank. Comparing to the speedups claimed for the low-rank assembling approach in [14], it appears that our speedups are roughly comparable for problems of similar rank. Of course, the exact numbers depend on implementation and hardware details.

We plot the times for the more difficult examples 2D-B and 3D-B in Figure 5. We display the cases $p = 2$ and $p = 4$ and again compare the full Gauss assembler and the proposed fast assembler. As expected, full Gauss assembly scales like $\mathcal{O}(n^d)$, since that is the scaling of the number of degrees of freedom for fixed p. Over the tested range of n, the fast assembler can be roughly estimated to scale like $\mathcal{O}(n^{1.4})$ in the 2D case and like $\mathcal{O}(n^{1.6})$ in the 3D case, thus behaving sublinearly in the number of degrees of freedom. The complexity analysis in

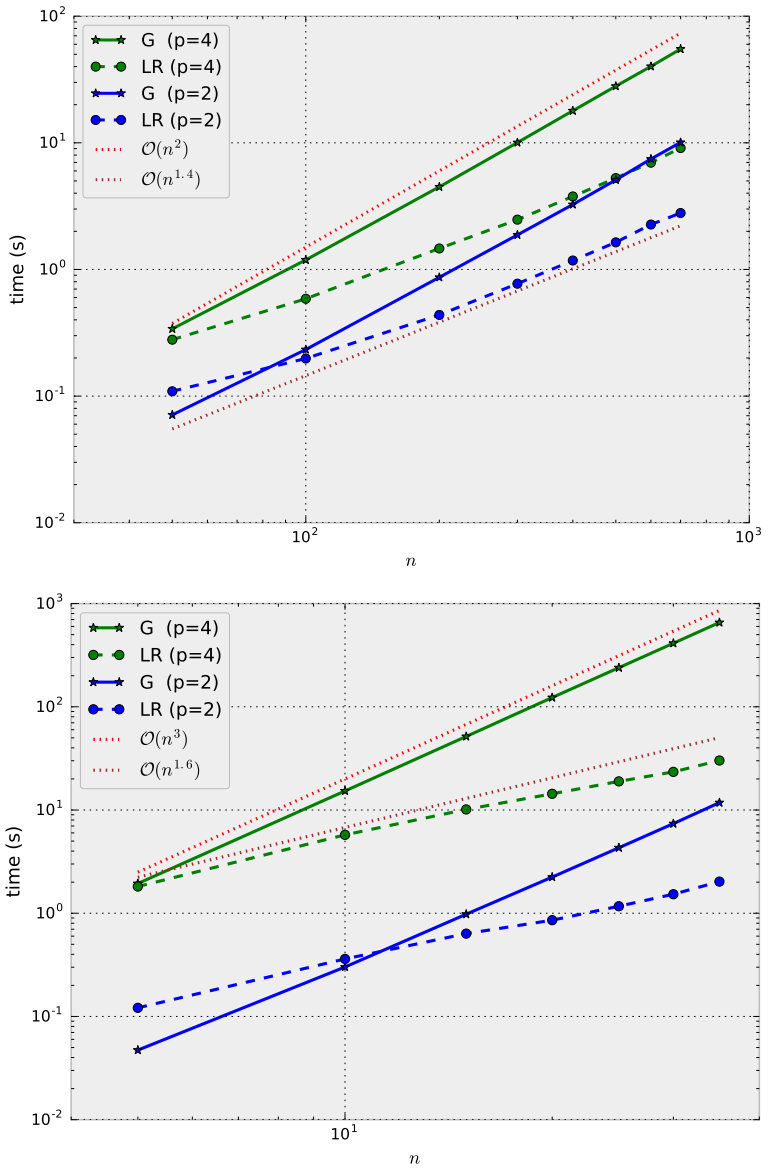


Figure 5: Timings for Examples 2D-B (top) and 3D-B (bottom), plotted over the number n of knot spans per coordinate direction. We compare the full Gauss assembler (solid lines labeled 'G') and our fast low-rank assembler (dashed lines labeled 'LR') for $p = 2$ and $p = 4$.

Sections 3.4 and 4.3 shows that asymptotically, the scaling has to approach $\mathcal{O}(n^d)$ for large n. This cannot be avoided as the method has to compute all the $\mathcal{O}(n^d p^d)$ nonzero entries of A. However, these examples show that there is a large pre-asymptotic regime where our fast assembler scales sublinearly.

References

[1] M. Bebendorf. Approximation of boundary element matrices. *Numerische Mathematik*, 86(4):565–589, 2000.

[2] M. Bebendorf. *Hierarchical Matrices*, volume 63 of *Lecture Notes in Computational Science and Engineering*. Springer Berlin Heidelberg, 2008.

[3] M. Bebendorf. Adaptive cross approximation of multivariate functions. *Constructive Approximation*, 34(2):149–179, 2011.

[4] I. Georgieva and C. Hofreither. An algorithm for low-rank approximation of bivariate functions using splines. *Journal of Computational and Applied Mathematics*, 310:80–91, 2017. Special Issue on Numerical Algorithms for Scientific and Engineering Applications.

[5] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, fourth edition, 2012.

[6] L. Grasedyck, D. Kressner, and C. Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013.

[7] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[8] W. Hackbusch, B. N. Khoromskij, and E. E. Tyrtyshnikov. Hierarchical Kronecker tensor-product approximations. *Journal of Numerical Mathematics*, 13(2):119–156, 2005.

[9] R.R. Hiemstra, F. Calabrò, D. Schillinger, and T.J.R. Hughes. Optimal and reduced quadrature rules for tensor product and hierarchically refined splines in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 316:966–1004, 2017. Special Issue on Isogeometric Analysis: Progress and Challenges.

[10] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39-41):4135–4195, October 2005.

[11] B.N. Khoromskij. Tensors-structured numerical methods in scientific computing: Survey on recent advances. *Chemometrics and Intelligent Laboratory Systems*, 110(1):1–19, 2012.

[12] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[13] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[14] A. Mantzaflaris, B. Jüttler, B. N. Khoromskij, and U. Langer. Low rank tensor methods in Galerkin-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 316:1062–1085, 2017. Special Issue on Isogeometric Analysis: Progress and Challenges.

[15] V. Olshevsky, I. Oseledets, and E. Tyrtyshnikov. Tensor properties of multilevel Toeplitz and related matrices. *Linear Algebra and its Applications*, 412(1):1–21, 2006.

[16] E. Tyrtyshnikov. Kronecker-product approximations for some function-related matrices. *Linear Algebra and its Applications*, 379:423–437, 2004.

[17] C. F. Van Loan and N. Pitsianis. Approximation with Kronecker products. In Marc S. Moonen, Gene H. Golub, and Bart L. R. De Moor, editors, *Linear Algebra for Large Scale and Real-Time Applications*, volume 232 of *NATO ASI Series*, pages 293–314. Springer Netherlands, Dordrecht, 1993.

n	p	1	2	3	4	5	6
100		0.09	0.22	0.54	1.14	2.56	5.52
200		0.31	0.83	1.98	4.49	10.05	21.87
300	Gauss	0.69	1.82	4.42	10.02	22.47	49.28
400	(seconds)	1.16	3.19	7.76	17.80	40.06	87.34
500		1.77	4.99	12.22	27.99	62.22	136.28
600		2.51	7.27	17.71	40.22	89.89	197.33
700		3.45	9.83	23.85	55.24	122.06	267.34
100		0.03	0.06	0.10	0.12	0.23	0.29
200		0.10	0.16	0.24	0.37	0.58	0.84
300	fast asm.	0.16	0.29	0.47	0.74	1.11	1.62
400	(seconds)	0.21	0.47	0.77	1.18	1.81	2.62
500		0.32	0.72	1.14	1.80	2.66	3.91
600		0.42	0.93	1.62	2.57	3.73	5.38
700		0.53	1.24	2.09	3.43	4.97	7.10
100		2.6	3.9	5.2	9.1	11.2	19.3
200		3.0	5.3	8.1	12.3	17.3	26.1
300		4.4	6.2	9.4	13.6	20.2	30.4
400	speedup	5.5	6.8	10.0	15.1	22.1	33.3
500		5.6	6.9	10.7	15.6	23.4	34.9
600		6.0	7.8	10.9	15.6	24.1	36.7
700		6.5	8.0	11.4	16.1	24.6	37.7
100		0.11	0.23	0.54	1.19	2.59	5.59
200		0.32	0.87	2.01	4.49	10.06	21.93
300	Gauss	0.68	1.87	4.49	10.04	22.57	49.54
400	(seconds)	1.19	3.26	7.89	17.91	40.16	87.85
500		1.81	5.09	12.39	28.00	62.38	137.00
600		2.58	7.45	17.87	40.14	90.14	197.01
700		3.49	10.08	24.12	55.06	122.72	269.49
100		0.12	0.20	0.34	0.59	1.13	2.06
200		0.24	0.44	0.80	1.46	2.64	4.39
300	fast asm.	0.38	0.77	1.45	2.47	4.47	7.81
400	(seconds)	0.55	1.18	2.11	3.77	6.64	11.11
500		0.76	1.64	3.09	5.27	9.07	15.00
600		1.01	2.26	4.24	6.96	11.96	19.64
700		1.29	2.79	5.32	9.10	15.37	24.12
100		0.9	1.2	1.6	2.0	2.3	2.7
200		1.4	2.0	2.5	3.1	3.8	5.0
300		1.8	2.4	3.1	4.1	5.0	6.3
400	speedup	2.1	2.8	3.7	4.7	6.0	7.9
500		2.4	3.1	4.0	5.3	6.9	9.1
600		2.6	3.3	4.2	5.8	7.5	10.0
700		2.7	3.6	4.5	6.1	8.0	11.2

Table 1: Results for Examples 2D-A (top) and 2D-B (bottom)

n	p	1	2	3	4
5	Gauss (seconds)	0.01	0.05	0.31	1.94
10		0.04	0.29	2.37	15.23
15		0.11	0.98	7.87	51.28
20		0.26	2.22	18.33	121.73
25		0.51	4.28	35.65	239.04
30		0.87	7.33	62.22	410.95
35		1.32	11.67	98.07	655.69
5	fast asm. (seconds)	0.01	0.01	0.03	0.09
10		0.01	0.03	0.12	0.24
15		0.02	0.08	0.20	0.48
20		0.05	0.12	0.35	0.90
25		0.06	0.22	0.61	1.49
30		0.09	0.37	1.03	2.27
35		0.14	0.60	1.47	3.46
5	speedup	1.4	4.1	10.2	22.4
10		3.3	11.7	20.6	63.6
15		5.4	11.8	39.7	106.4
20		4.8	18.7	52.6	135.4
25		8.4	19.9	58.9	160.1
30		9.3	19.6	60.6	181.3
35		9.6	19.5	66.5	189.6
5	Gauss (seconds)	0.01	0.05	0.32	1.94
10		0.04	0.30	2.37	15.29
15		0.11	0.98	7.88	51.57
20		0.26	2.24	18.37	122.86
25		0.52	4.31	36.03	239.15
30		0.87	7.38	62.57	412.52
35		1.34	11.79	98.28	655.12
5	fast asm. (seconds)	0.02	0.12	0.53	1.81
10		0.09	0.36	1.70	5.75
15		0.15	0.63	2.57	10.13
20		0.22	0.86	3.87	14.35
25		0.31	1.17	5.06	18.94
30		0.40	1.53	6.83	23.40
35		0.50	2.03	8.64	30.17
5	speedup	0.4	0.4	0.6	1.1
10		0.4	0.8	1.4	2.7
15		0.8	1.5	3.1	5.1
20		1.2	2.6	4.7	8.6
25		1.7	3.7	7.1	12.6
30		2.2	4.8	9.2	17.6
35		2.7	5.8	11.4	21.7

Table 2: Results for Examples 3D-A (top) and 3D-B (bottom)