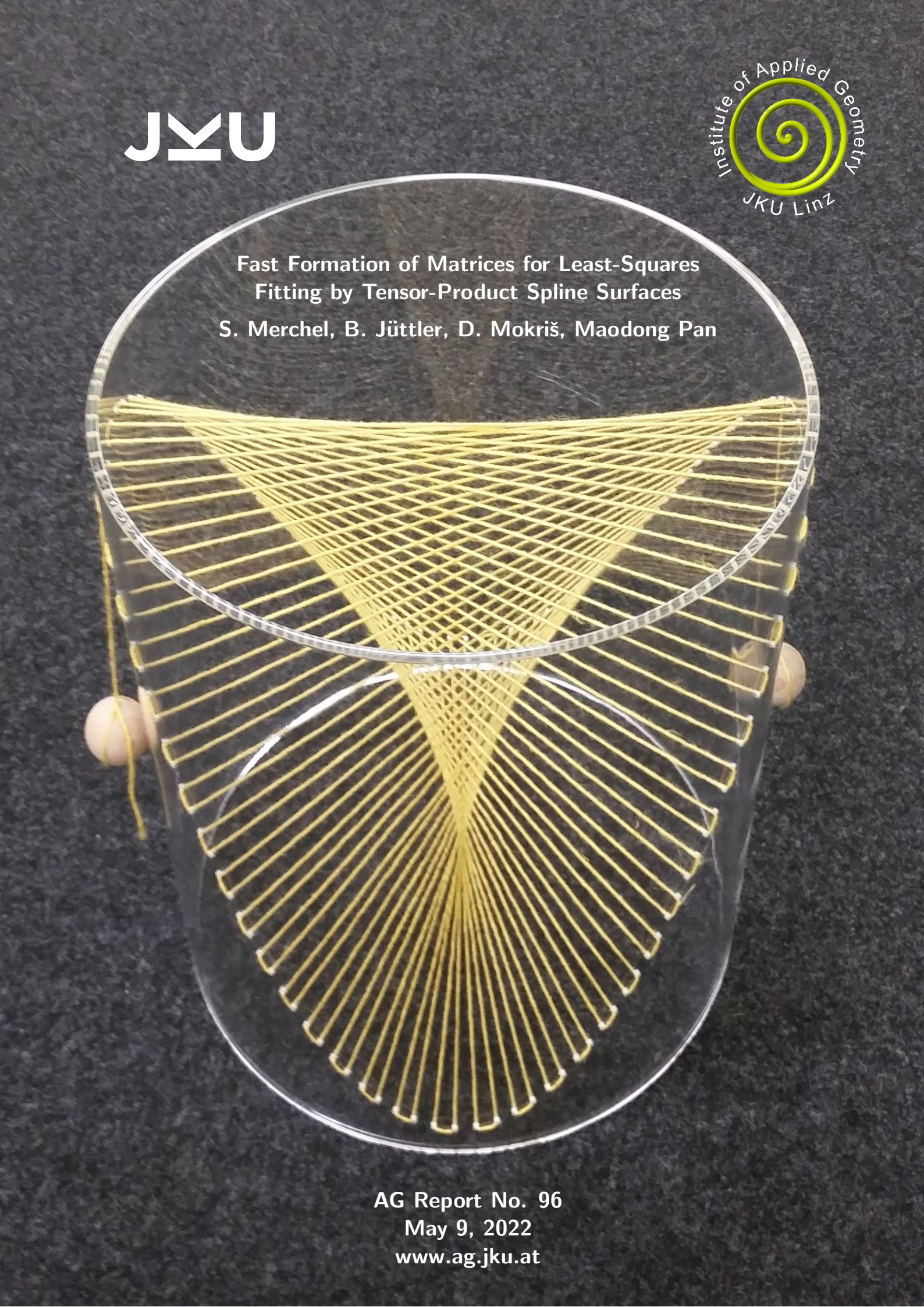


Fast Formation of Matrices for Least-Squares Fitting by Tensor-Product Spline Surfaces

S. Merchel, B. Jüttler, D. Mokriš, Maodong Pan

AG Report No. 96
May 9, 2022
www.ag.jku.at

Fast Formation of Matrices for Least-Squares Fitting by Tensor-Product Spline Surfaces

Sandra Merchel[a], Bert Jüttler[b], Dominik Mokriš[a], Maodong Pan[c,*]

[a]*MTU Aero Engines AG, Munich, Germany*
[b]*Institute of Applied Geometry, Johannes Kepler University Linz, Austria*
[c]*School of Mathematics, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China*

Abstract

Least-squares fitting by tensor-product spline surfaces is a classical method for approximating unstructured data, which is widely used in industry. However, assembling the system of equations via the straightforward approach can be quite time–consuming. In this paper, we propose to accelerate this process by employing the technique of sum factorization, which is frequently used in the context of isogeometric analysis.

Our approach consists of two steps. First, we introduce a regular grid onto which the parameters of the data are projected. Consequently, the expressions of the matrix entries take a form that admits the use of sum factorization, which is then employed in the second step. We provide a detailed complexity analysis and quantify the expected relative assembly costs. Several examples, including an example involving industrial data, demonstrate how the choice of the grid influences speed and precision, and confirm the expected time savings of the proposed method.

Keywords: least-squares fitting; scattered data approximation; matrix formation; sum factorization; tensor-product splines; computational cost.

1. Introduction

The task of creating mathematical representations of free-form surfaces from scattered data is an important and therefore well-studied problem, which is particularly challenging and useful in industrial applications. It has attracted the attention from numerous researchers over the years. The fundamentals of *least-squares approximation by tensor-product spline surfaces* are described in the classical textbook of Hoschek and Lasser [1]. We briefly list some of the contributions to this topic.

In order to make this process more efficient, Cheng and Goshtasby proposed to split the problem into several curve fitting problems, see [2], thereby exploiting the tensor-product structure. Even though this reduces the computational effort, it is applicable only if the parameter values of the data form a regular grid, which is usually not the case for industrial data. Other extensions of the original framework make use of adaptive spline refinement, which has been pioneered by Forsey and Bartels [3]. Various advanced techniques for least-squares surface fitting were summarized by Weiss et al. [4].

Krishnamurthy and Levoy [5] described a method for converting dense polygon meshes into tensor-product surfaces plus associated displacement maps, in order to facilitate animation and rendering. Simultaneously, Eck et al. [6] showed how to deal with data of arbitrary topology, approximating them by a collection of tensor-product B-spline patches. Greiner and Hormann [7] also discussed surface reconstruction in the context of scattered data fitting, focusing on techniques for representing details via hierarchical spline refinement, and for improving the fairness of the surface. Various techniques for geometry reconstruction from scattered data – a process which is sometimes called "reverse engineering" – are described in the survey article by Várady and Martin [8].

The parameterization of the data is a critical task in the context of data fitting. Floater [9] (see also [10]) established powerful and easy-to-use methods to assign parameter values to unstructured sets of data, which are suitable for surface approximation and widely used in practice. Methods for optimizing the parameterization have attracted the attention of numerous researchers, see e.g. [11–13].

Further topics that have been discussed include constructions for convex surfaces [14], lofted B-spline surfaces [15], and fitting with adaptive refinement via truncated hierarchical B-splines [16, 17]. Recent contributions address extensions to T-spline surfaces [18–20] and surface fitting via the method of progressive iterative approximation [21, 22], which is closely related to the gradient descent method as noted in [23].

In order to improve the efficiency of methods for surface reconstruction from scattered data, we propose to combine the mathematical technology of least-squares approximation by tensor-product spline surfaces with the use of *sum factorization* for fast matrix formation. This technique was originally proposed for assembling the matrices that arise when performing numerical simulation via spectral methods [24]. More recently, it has been successfully applied in the context of isogeometric Galerkin discretizations [25–31].

The initial paper [25] performed the assembly process in an element-wise way. Later, Bressan and Takacs [27] improved this work by introducing a global variant of sum factorization, which dramatically reduces the computational cost.

An efficient assembling approach was proposed in [26] by use of weighted quadrature and sum factorization techniques. The application of this method to isogeometric linear elasticity [28] was also explored. However, this approach achieves its efficiency by sacrificing the symmetry of the resulting matrices.

In order to address this issue, Pan and Jüttler [29] introduced a fast matrix assembly algorithm based on the interpolation, look-

*Corresponding author.
Email addresses: sandra.merchel@mtu.de (Sandra Merchel), bert.juettler@jku.at (Bert Jüttler), dominik.mokris@mtu.de (Dominik Mokriš), mdpan@mail.ustc.edu.cn (Maodong Pan)

up and sum factorization techniques. This algorithm is competitive with the previous method [26] in terms of the computational cost, while preserving the symmetry of the system matrices.

More recently, the use of sum factorization has been further extended to the case of bivariate hierarchical B-splines [30] and later even to hierarchical B-splines in any dimension [31].

Since matrix formation requires a substantial part of the overall computation time of scattered data approximation by tensor-product spline surfaces, we propose to extend the applicability of sum factorization to least-squares fitting with tensor-product spline surfaces. The application of sum factorization is not straightforward in this context, since we initially do not have the required structure of nested sums in the equations governing the matrix elements appearing in the linear system for the least-squares approximation problem. Therefore, we first need to project the parameters of the data onto a regular grid to achieve this type of formulation, and are consequently able to perform sum factorization. This novel approach has the potential of significantly reducing the computational effort of the matrix assembly and therefore the total cost of the least-squares approximation.

The paper is structured as follows: We start with introducing the standard least-squares fitting and derive the equivalent linear system of equations. The straightforward matrix assembly method is presented and its costs are analyzed. Section 3 shows how to transform the linear system into a form that is suitable for the use of sum factorization, by projecting the parameters of the data onto a regular grid. After applying sum factorization, the algorithm of the resulting fast matrix assembly is presented. This is followed by a detailed analysis of the computational complexity. Based on this analysis we estimate the relative assembly costs of our method compared to the standard method. Section 4 presents several examples that demonstrate the influence of the grid projection on the precision and speed for several grid choices. Furthermore we compare the convergence rate of the new method to the standard method for data without and with noise. We also use an example from industry to underline the efficiency and usefulness of the new method. Finally we conclude the paper and discuss future work.

2. Preliminaries

We consider two bases $\{\beta_i(u)\}_{i=1}^m$ and $\{\gamma_j(v)\}_{j=1}^n$ of univariate B-splines of degree d on the interval $[0,1]$. We use these spline bases to define the tensor-product spline surface

$$\mathbf{s}(u,v) = \sum_{i=1}^{m}\sum_{j=1}^{n} \mathbf{b}_{ij}\beta_i(u)\gamma_j(v)$$

of degree (d,d) with control points $\mathbf{b}_{ij} \in \mathbb{R}^3$. These control points (which are represented by row vectors) are collected in a matrix $\mathbf{b}$ of dimension $mn \times 3$. The second index corresponds to the xyz coordinates, while the first one is used for enumerating of control points, which is done lexicographically with respect to (i,j).

The problem we are interested in can be stated as follows: Given the point data $\{\mathbf{f}_k\}_{k=1}^N$ in $\mathbb{R}^3$, our task is to find the associated parameters $\{(u_k,v_k)\}_{k=1}^N$ in $[0,1]^2$ and the control points $\mathbf{b}$ such that the least-squares sum

$$\sum_{k=1}^{N} \|\mathbf{s}(u_k,v_k) - \mathbf{f}_k\|^2 + R(\mathbf{b}) \tag{1}$$

is minimized, where $R(\mathbf{b})$ might be an additional regularization term. The established method for this task proceeds in two steps:

- First, we assign the parameter values $(u_k,v_k) \in [0,1]^2$ to the given data $\mathbf{f}_k$, $k = 1,\dots,N$. This can be done by using the Floater's parameterization method, which is well established in the literature [9].
- Second, the control points $\mathbf{b}$ are obtained by minimizing the least-squares sum (1), which leads to the linear system

$$M\mathbf{b} = \mathbf{r} \tag{2}$$

with a sparse and banded matrix M of size $mn \times mn$ and right-hand-side matrix $\mathbf{r}$ of size $mn \times 3$.

The elements of M take the form

$$M_{(i,j)(i',j')} = \sum_{k=1}^{N} \beta_i(u_k)\gamma_j(v_k)\beta_{i'}(u_k)\gamma_{j'}(v_k), \tag{3}$$

if no regularization is used. The mn row vectors of the right-hand side can be written as

$$r_{(i,j)} = \sum_{k=1}^{N} \beta_i(u_k)\gamma_j(v_k)\mathbf{f}_k. \tag{4}$$

The linear equations (2) can be solved with the help of direct or iterative solvers.

We are interested in algorithms that generate the elements of the system matrix M. The standard approach to compute the matrix elements appearing in (3) and (4) proceeds by looping over all the data points, and then evaluating the nonzero basis functions at each point. The resulting procedure, which we call the *standard algorithm for matrix assembly* for least-squares fitting, is summarized in the following algorithm:

Standard algorithm for matrix assembly

```
for i = 1 to m do
    for j = 1 to n do
        for i' = 1 to m do
            for j' = 1 to n do
                M_{(i,j)(i',j')} = 0                          ▷ initialization
for k = 1 to N do
    Evaluate the d + 1 active basis functions β_{i''} at the point u_k
    Evaluate the d + 1 active basis functions γ_{j''} at the point v_k
    for i ∈ B(u_k) do                                         ▷ β_i(u_k) ≠ 0
        K_1 = β_i(u_k)
        for j ∈ G(v_k) do                                     ▷ γ_j(v_k) ≠ 0
            K_2 = K_1 γ_j(v_k)
            for i' ∈ B(u_k) do
                K_3 = K_2 β_{i'}(u_k)
                for j' ∈ G(v_k) do
                    M_{(i,j)(i',j')} += K_3 γ_{j'}(v_k)       ▷ summation
```

The index sets $\mathcal{B}(u_k)$ and $\mathcal{G}(v_k)$ are defined by

$$\mathcal{B}(u_k) = \{i' | \beta_{i'}(u_k) \neq 0\} \text{ and } \mathcal{G}(v_k) = \{j' | \gamma_{j'}(v_k) \neq 0\},$$

respectively, where we note that the sizes of the index sets satisfy

$$|\mathcal{B}(u_k)| \leq d+1 \text{ and } |\mathcal{G}(v_k)| \leq d+1\ .$$

In fact, they are equal to $d+1$ unless u_k or v_k is a knot.

Proposition 1. *The standard algorithm takes*

$$N(d+1)^2(2d^2+5d+4) \tag{5}$$

floating point operations (flops), and additionally $2N(d+1)$ *evaluations of univariate B-splines to assemble the matrix* M.

Proof. The evaluation of the matrix elements proceeds by looping over all the parameters $\{u_k, v_k\}_{k=1}^N$:

1. For each parameter, we perform $2(d+1)$ evaluations of univariate B-splines and visit $(d+1)^2$ index pairs (i_1, j_1).
2. For each pair (i_1, j_1), the calculation of K_2 requires one flop, and $d+1$ instances of the index i' are visited.
3. For each i', the evaluation of K_3 needs one flop, and $d+1$ instances of the index j' are visited.
4. Finally we compute the element $M_{(i,j),(i',j')}$ for each j'. It needs 2 flops per loop.

Summing up, assembling the matrix M via the above standard procedure needs

$$N(d+1)^2(1+(d+1)(1+(d+1)2)) = N(d+1)^2(2d^2+5d+4)$$

flops and

$$2N(d+1)$$

evaluations of univariate B-splines. □

Here we only discuss the matrix M, since the right-hand-side one can be dealt with analogously. We note that the cost of assembling the matrices M and $\mathbf{r}$ via the standard algorithm constitutes a significant portion of the overall computational effort. Therefore, we focus on deriving a faster method for the assembly process.

3. Fast matrix assembly

In order to improve the computational efficiency, we apply the sum factorization technique to the process of matrix assembly. In order to enable this approach, we introduce auxiliary *gridded* parameters and replace the original ones by a suitable subset of those. On the one hand, this allows us to accelerate the matrix assembly, since we may systematically re-use some of the computed quantities. On the other hand, this may compromise the quality of the fitting result, since we modify the parameterization of the data.

This section describes the approach in more detail and focuses on the first aspect. The second effect will be analyzed in the subsequent section, based on numerical experiments.

3.1. The algorithm

In order to enable the sum factorization, we introduce the *gridded* parameters

$$(\hat{u}_p, \hat{v}_q), \quad p = 1, \dots, \hat{m}, \quad q = 1, \dots, \hat{n},$$

which form a grid of size $\hat{m} \times \hat{n}$. Each data parameter (u_k, v_k) has the associated gridded parameter

$$(\hat{u}_{p_k}, \hat{v}_{q_k}), \quad k = 1, \dots, N,$$

which is its nearest point among all the gridded parameters. Thus, the index pair

$$(p_k, q_k)$$

identifies the grid point, which is associated to the data point with index k.

Let μ_{pq} be the number count (multiplicity) of data points that are associated with the given grid point $(\hat{u}_p, \hat{v}_q)$. We rewrite the matrix elements $M_{(i,j)(i',j')}$ as

$$\begin{aligned} M_{(i,j)(i',j')} \approx \hat{M}_{(i,j)(i',j')} &= \sum_{k=1}^{N} \beta_i(\hat{u}_{p_k})\gamma_j(\hat{v}_{q_k})\beta_{i'}(\hat{u}_{p_k})\gamma_{j'}(\hat{v}_{q_k}) \\ &= \sum_{p=1}^{\hat{m}} \sum_{q=1}^{\hat{n}} \mu_{pq}\beta_i(\hat{u}_p)\gamma_j(\hat{v}_q)\beta_{i'}(\hat{u}_p)\gamma_{j'}(\hat{v}_q). \end{aligned}$$

Note that μ_{pq} is zero if no data points are assigned to the grid parameter $(\hat{u}_p, \hat{v}_q)$.

When applied in assembling the matrices arising from isogeometric discretizations, the technique of sum factorization achieves its efficiency by exploiting the tensor-product structure of multivariate B-splines. Using this approach in our case, the matrix elements $\hat{M}_{(i,j)(i',j')}$ take the form

$$\hat{M}_{(i,j)(i',j')} = \sum_{p=1}^{\hat{m}} \beta_i(\hat{u}_p)\beta_{i'}(\hat{u}_p) \underbrace{\sum_{q=1}^{\hat{n}} \mu_{pq}\gamma_j(\hat{v}_q)\gamma_{j'}(\hat{v}_q)}_{= H_{jj'p}}, \tag{6}$$

where we introduce the auxiliary tensor $\boldsymbol{H} = (H_{jj'p})$ of order 3.

All the elements $\hat{M}_{(i,j)(i',j')}$ in (6) are calculated in a recursive way: The first stage of the recursion evaluates and stores all the nonzero elements of the tensor $\boldsymbol{H}$, which are used for computing $\hat{M}_{(i,j)(i',j')}$ in the second stage. The detailed description of the new assembly algorithm is summarized in the following algorithm, which consists of two parts.

The first one generates the auxiliary tensor $\boldsymbol{H} = (H_{jj'p})$:

Fast algorithm for matrix assembly – part 1

Create the tensor $\boldsymbol{H}$ and initialize its elements by zero
for $k = 1$ to N **do**
  Evaluate the $d+1$ active (i.e., nonzero) basis functions $\gamma_{j''}$ at $\hat{v}_{q_k}$
  for $j \in \mathcal{G}(\hat{v}_{q_k})$ **do** ▷ $\gamma_j(\hat{v}_{q_k}) \neq 0$
    for $j' \in \mathcal{G}(\hat{v}_{q_k})$ **do** ▷ $\gamma_{j'}(\hat{v}_{q_k}) \neq 0$
      $H_{jj'p_k} \mathrel{+}= \gamma_j(\hat{v}_{q_k})\gamma_{j'}(\hat{v}_{q_k})$ ▷ summation

The second part performs the evaluation of the matrix elements:

Fast algorithm for matrix assembly – part 2

for $i = 1$ to m **do**
  for $j = 1$ to n **do**
    for $i' \in \mathbb{B}(i)$ **do** ▷ $\operatorname{supp}(\beta_i\beta_{i'}) \neq \emptyset$
      for $j' \in \mathbb{G}(j)$ **do** ▷ $\operatorname{supp}(\gamma_j\gamma_{j'}) \neq \emptyset$
        $\hat{M}_{(i,j)(i',j')} = 0$ ▷ initialization
for $p = 1$ to $\hat{m}$ **do**
  Evaluate the $d+1$ active basis functions $\beta_{i''}$ at the point $\hat{u}_p$
  for $j = 1$ to n **do**
    for $j' \in \mathbb{G}(j)$ **do** ▷ $\operatorname{supp}(\gamma_j\gamma_{j'}) \neq \emptyset$
      if $H_{jj'p} > 0$ **then**
        for $i \in \mathcal{B}(\hat{u}_p)$ **do** ▷ $\beta_i(\hat{u}_p) \neq 0$
          $K = \beta_i(\hat{u}_p)H_{jj'p}$
          for $i' \in \mathcal{B}(\hat{u}_p)$ **do** ▷ $\beta_{i'}(\hat{u}_p) \neq 0$
            $\hat{M}_{(i,j)(i',j')} \mathrel{+}= \beta_{i'}(\hat{u}_p)K$ ▷ summation

The index sets $\mathcal{B}(\hat{u}_p)$ and $\mathcal{G}(\hat{v}_{q_k})$ are defined as before. The new index sets $\mathbb{B}(i)$ and $\mathbb{G}(j)$ are given by

$$\mathbb{B}(i) = \{i' | \operatorname{supp}\beta_i\beta_{i'} \neq \emptyset\} \text{ and } \mathbb{G}(j) = \{j' | \operatorname{supp}\gamma_j\gamma_{j'} \neq \emptyset\},$$

respectively. We note that the sizes of these index sets are bounded by

$$|\mathbb{B}(i)| \leq 2d+1, \ |\mathbb{G}(j)| \leq 2d+1.$$

3.2. Complexity analysis

The proposed approach for least-squares fitting consists of two steps: projecting each data parameter to the associated gridded parameter and assembling the matrices via sum factorization.

The cost of the first step is negligible compared to the overall complexity of the new method. The second step of the proposed approach assembles the elements of the matrices and consists of the two parts as described above.

The first part, which computes the auxiliary tensor $\boldsymbol{H}$, calculates at most $N(d+1)^2$ nonzero elements $H_{jj'p_k}$ by summing over $N(d+1)^2$ index triplets (k, j, j') and needs 2 flops per term. This amounts to

$$2N(d+1)^2$$

flops. In addition, it requires

$$(d+1)\min(N, \hat{n})$$

evaluations of univariate B-splines.

The second part, which computes the matrix elements $\hat{M}_{(i,j),(i',j')}$, proceeds by iterating over the nonzero elements $H_{jj'p}$ and the index pairs (i, i'), requiring two flops per term. It also generates the intermediate elements K used for subsequent evaluations, which needs $d+1$ flops per nonzero element $H_{jj'p}$. Moreover, this loop also performs

$$(d+1)\min(N, \hat{m})$$

evaluations of univariate B-splines.

In order to estimate the cost of the second part, we need an estimate of how many nonzero elements $H_{jj'p}$ exist. In order to derive this estimate, we make the following assumption:

> *We assume that the parameters of the data are random variables with values in* $[0,1]^2$*, which are uniformly distributed.*

Moreover, we restrict the analysis to the case of *uniform knot vectors with single knots*.

We consider the probability

$$\xi_{|j-j'|} = P(H_{jj'p} = 0)\,,$$

and note that it is independent of p and depends solely on the absolute value of the index difference $j - j'$, if one does not take the effects of the patch boundary into account (as we shall do from now on).

Next, we note that $H_{jj'p} = 0$ exactly if

$$\sum_{q=1}^{\hat{n}} \mu_{pq}\gamma_j(\hat{v}_q)\gamma_{j'}(\hat{v}_q) = 0\,,$$

i.e., exactly if the overlapping domain of the supports of γ_j and $\gamma_{j'}$ in row p do not contain any gridded data. The overlap has length $\max(\frac{d+1-\ell}{n}, 0)$ without taking boundary effects into account. Consequently, we evaluate ξ_ℓ as

$$\xi_\ell = \begin{cases} \left(1 - \frac{d+1-\ell}{\hat{m}n}\right)^N & \text{if } 0 \le \ell \le d\,, \\ 1 \text{ if } \ell > d\,, \end{cases} \tag{7}$$

which is the probability for a random distribution of N data points not to possess any instance in a box of width $\max(\frac{d+1-\ell}{n}, 0)$ and height $\frac{1}{\hat{m}}$ (that corresponds to the considered row p of the grid).

We use these results to evaluate the expected costs of the second loop in part 2 of the fast matrix assembly and arrive at

$$\Big(\sum_{p=1}^{\hat{m}}\sum_{j=1}^{n}\sum_{j'\in\mathbb{G}(j)}(1-\xi_{|j-j'|})\Big)(d+1)(1+2(d+1)) \tag{8}$$

flops.

In order to analyze the effect of the random distribution of the parameter values, we introduce the auxiliary quantity ξ_{avg} via the identity

$$\begin{aligned}(2d+1)\,\hat{m}n\,\xi_{\text{avg}} &= \sum_{p=1}^{\hat{m}}\sum_{j=1}^{n}\sum_{j'\in\mathbb{G}(j)}\xi_{|j-j'|} \\ &= \sum_{k=0}^{d}\underbrace{\sum_{p=1}^{\hat{m}}\sum_{j,j'\,:\,|j-j'|=\ell}}_{(\star)}\xi_\ell\ .\end{aligned} \tag{9}$$

This quantity ξ_{avg} describes the expected share of the cases where the condition of the "if" statement in the algorithm is not satisfied. Indeed, we consider

$$(2d+1)\hat{m}n$$

terms (ignoring boundary effects) in the sums that correspond to the loops of the algorithm.

Note that the summands ξ_ℓ in $(\star)$ in Eq. (9) are independent of the summation indices p, j, j'. We count the number of these summands,

$$\begin{cases} 2\hat{m}n & \text{if } 1 \le \ell \le d \\ \hat{m}n & \text{if } \ell = 0\,, \end{cases}$$

and this allows us to arrive at the formula

$$\xi_{\text{avg}} = \frac{1}{2d+1}(\xi_0 + 2\sum_{\ell=1}^{d}\xi_\ell)\,. \tag{10}$$

Finally we use this result to rewrite (8) with the help of (9). This confirms that the expected computational costs of part 2 amount to

$$(2d+1)\hat{m}n(1-\xi_{\text{avg}})(d+1)(1+2(d+1))$$

flops and

$$(d+1)\min(N, \hat{m})$$

evaluations of univariate B-splines. Summing up, if we do not consider effects caused by the patch boundaries, we obtain the following result:

Proposition 2. *The fast algorithm for matrix assembly is expected to take*

$$2N(d+1)^2 + (1-\xi_{\text{avg}})(d+1)(2d+1)(2d+3)\hat{m}n \tag{11}$$

flops, where ξ_{avg} *is given by Eq.* (10)*, and*

$$(d+1)(\min(N, \hat{m}) + \min(N, \hat{n}))$$

evaluations of univariate B-splines to generate the matrix M.

For instance, the expected number of flops evaluates to

$$32N + 252(1-\xi_{\text{avg}})\hat{m}n$$

with

$$\xi_{\text{avg}} = \frac{(\hat{m}n-4)^N + 2(\hat{m}n-3)^N + 2(\hat{m}n-2)^N + 2(\hat{m}n-1)^N}{7(\hat{m}n)^N}$$

for bicubic spline surfaces ($d = 3$).

3.3. Expected relative assembly costs as $N \to \infty$

In order to analyze the behaviour of the quantity ξ_{avg} and to quantify the expected relative assembly costs of the fast matrix assembly procedure with respect to the standard approach we shall consider the limit $N \to \infty$. Moreover, we introduce the three quantities Δ, δ and ϱ, which are defined by

$$\Delta = \frac{mn}{N},\ \delta = \frac{N}{\hat{m}\hat{n}}, \quad \text{and} \quad \varrho = \frac{\delta}{\Delta}$$

respectively. The first and the second one quantify the number of control points per data point and the number of data points per grid point, respectively. These two quantities are expected to be significantly smaller than 1. The third quantity ϱ is the *double ratio* of the two ratios.

Well-known identities can be invoked to confirm

$$\lim_{N\to\infty}\left(1-\frac{C}{\hat{m}n}\right)^N = \lim_{N\to\infty}\left(1-\frac{C}{\sqrt{\frac{N}{\delta}}\sqrt{N\Delta}}\right)^N = e^{-\frac{C\sqrt{\delta}}{\sqrt{\Delta}}} = e^{-C\sqrt{\varrho}},$$

which gives

$$\Xi(d,\varrho) = \lim_{N\to\infty}\xi_{\text{avg}} = \frac{1}{2d+1}\left(e^{-(d+1)\sqrt{\varrho}} + 2\sum_{k=1}^{d} e^{-k\sqrt{\varrho}}\right). \quad (12)$$

We use this observation to quantify the expected value of the relative assembly costs (which is obtained as the ratio of the numbers of required flops of the two algorithms) for large data sets. Here we do not take the number of B-spline evaluations into account, since the total effort is dominated by the number of flops. Moreover, the fast method needs at most the same number of evaluations as the standard approach.

Theorem 3. *The expected value Ψ of the relative assembly costs of the fast matrix assembly with respect to the standard method takes the value*

$$\Psi(d,\varrho) = \frac{2(d+1) + \dfrac{1-\Xi(d,\varrho)}{\sqrt{\varrho}}(2d+1)(2d+3)}{(d+1)(2d^2+5d+4)}, \quad (13)$$

as $N \to \infty$, where $\Xi(d,\varrho)$ is defined in (12)*. We note that this quantity depends solely on the degree d and the double ratio ϱ.*

Proof. We use (5) and (11) from Proposition 1 and Proposition 2 to evaluate the quotient of the numbers of required flops, and we simplify it with the help of the quantities Δ, δ and ϱ. □

As an example, we evaluate the expected value of the relative assembly costs of the cubic splines ($d = 3$) and get

$$\frac{2}{37} + \left(1 - \frac{1}{7}\left(e^{-4\sqrt{\varrho}} + 2\sum_{k=1}^{3} e^{-k\sqrt{\varrho}}\right)\right)\frac{1}{\sqrt{\varrho}}\frac{63}{148}.$$

We list a few values of this quantity:

ϱ	0.25	0.5	1.0	3.0	9.0
Ψ	59.77%	50.37%	41.14%	28.48%	19.38%

Table 1: Expected relative assembly costs for $d = 3$ and different values of ϱ

3.4. The case of sparsely populated grids

In this situation, the number of grid points is very large compared to the number of data points. This can be seen as using the original parameterization of the data points, where the quality of the fitting result is not affected by the process of matrix assembly. Indeed, the use of floating point numbers automatically imposes a grid, although a very fine one. The scenario corresponds to the limit

$$\varrho \to 0\,.$$

We use de l'Hôpital's rule to confirm the identity

$$\lim_{\varrho\to 0}\frac{1-\Xi(d,\varrho)}{\sqrt{\varrho}} = \frac{(d+1)^2}{2d+1}$$

and use it to evaluate the limit of the expected relative assembly costs as

$$\lim_{\varrho\to 0}\Psi(d,\varrho) = 1 + \frac{1}{2d^2+5d+4}\,.$$

While it is slightly larger than 1 – which means that the fast method is slower than the standard one in the worst case – the costs of both approaches are virtually identical in this scenario. Considering low degrees $d = 2, 3, 4$, this ratio is equal to

$$\frac{23}{22},\ \frac{38}{37}, \quad \text{and} \quad \frac{57}{56}\,,$$

respectively. We conclude that the use of the fast assembly algorithm does not incur significant additional costs even in the limit case of sparsely populated grids.

4. Numerical results

We present six examples, which support the theoretical findings regarding the computational complexity and demonstrate the influence of the gridding on the quality of the resulting surface.

4.1. Accuracy of ξ_{avg}, Ξ and Ψ

In the *first example*, we compare the estimates for ξ_{avg} and Ξ (see Equations (10) and (12)) with the actual share of the cases where the condition of the "if" statement in the algorithm is not satisfied. We consider four sets of randomly generated parameter values (which are uniformly distributed in $[0,1]^2$) of different sizes N and various values of degrees of freedom and of the grid size, all for cubic splines ($d = 3$). For simplicity, we choose symmetric grids, $m = n$ and $\hat{m} = \hat{n}$.

Table 2 reports the observed share of the cases and compares it with the predicted share ξ_{avg} and the expected limit Ξ of that share for $N \to \infty$. We see that the estimates are fairly accurate. (The values of ξ_{avg} and Ξ are actually different, but they share the first digits.) As expected, the deviation is larger for smaller instances of data sets, since the estimates do not take boundary effects into account.

Δ	δ	N	ϱ	ξ_{avg}	Ξ	actual share
0.01	0.0625	122,500	6.25	2.55e−2	2.55e−2	2.23e−2
0.01	0.0625	4,900	6.25	2.55e−2	2.55e−2	4.92e−3
0.01	0.01	122,500	1.0	1.61e−1	1.61e−1	1.50e−1
0.01	0.01	36,100	1.0	1.61e−1	1.61e−1	1.42e−1
0.0361	0.128	10,000	3.53	5.13e−2	5.14e−2	4.60e−2
0.01	0.0025	122,500	0.25	3.61e−1	3.61e−1	3.48e−1

Table 2: First example – estimates for ξ_{avg} and Ξ and actual share of cases

In the *second example*, we consider a single data set with $N = 122,500$ and $\Delta = 0.01$ (i.e., $m = n = 35$) for various values of the grid size $\hat{m} = \hat{n}$, again for $d = 3$. We compare the actual and the estimated relative assembly costs, based on the number of flops needed to assemble the matrix, including evaluations of the basis functions. The differences between the observed numbers and the estimates are due to boundary effects and also caused by the cost of the evaluations, which are not considered for Ψ. The results, which are reported in Table 3, indicate a good agreement between the theoretically predicted and the experimentally observed numbers.

	relative assembly costs	
ϱ	Ψ	experimentally observed
1.0	41.1%	35.8%
2.78	29.2%	25.4%
6.25	22.0%	19.1%

Table 3: Second example – expected and numerically observed relative assembly costs for different values of the double ratio ϱ.

4.2. The trade-off between accuracy and computational costs

Clearly, one expects to see a trade-off between accuracy and computational costs, as the use of large grids eliminates the computational advantage of the fast assembly algorithm, while large values of δ will compromise the quality of the fitting result. This effect is demonstrated in the *third example*.

We consider $N = 40,000$ points, which are randomly sampled (i.e., using random variables with a uniform distribution in $[0,1]^2$) from the graph surface of the function

$$f(u,v) = \frac{1}{3}\sin(4\pi u)\sin(4\pi v)\,,$$

and approximate them using a bicubic spline surface with

$$n \times m = 20 \times 20$$

control points, i.e., we get $\Delta = 0.01$. Table 4 reports the mean square error and the relative assembly costs (both with respect to the number of flops and the actual computation times) for various values of the grid size.

We use the mean square error ("MSE/def") defined by

$$\text{MSE/def} = \frac{1}{N}\sum_{k=1}^{N} \|\mathbf{s}(u_k, v_k) - \mathbf{f}_k\|^2,$$

whereas the mean square error with use of parameter correction ("MSE/pc") is defined as

$$\text{MSE/pc} = \frac{1}{N}\sum_{k=1}^{N} \min_{(u,v)\in[0,1]^2} \|\mathbf{s}(u,v) - \mathbf{f}_k\|^2.$$

$\hat{m} = \hat{n}$	200	400	800	1600	3200	6400	standard
δ	1	0.25	$6.25e{-2}$	$1.56e{-2}$	$3.91e{-3}$	$9.77e{-4}$	0
MSE/def	$4.8e{-4}$	$2.4e{-4}$	$1.4e{-4}$	$9.9e{-5}$	$8.6e{-5}$	$8.1e{-5}$	$7.9e{-5}$
MSE/pc	$9.5e{-5}$	$5.5e{-5}$	$4.1e{-5}$	$3.6e{-5}$	$3.4e{-5}$	$3.4e{-5}$	$3.4e{-5}$
flops	8.34%	11.86%	18.66%	30.31%	46.48%	63.37%	100%
time	3.85%	6.03%	10.00%	17.48%	28.25%	38.01%	100%

Table 4: Third example – mean square error (MSE) and share of the computational costs for various values of δ and $d = 3$.

Two observations are in order:

First, the use of parameter correction ("pc") significantly decreases the measured error, compared to the default ("def") way of evaluating the MSE. In particular, this difference is significant when using the fast method with coarser grids.

Second, the expected trade-off between computational effort and accuracy is clearly present and visualized in Fig. 1, left. The choice of δ possesses an impact both on the error and on the costs of our method. In the considered setting, choosing a value in the interval

$$\delta \in [0.00391, 0.0625]$$

seems to be reasonable as it decreases the computational costs significantly without too much impact on the approximation quality of the result.

We carry out the same experiment for degree $d = 5$ and again observe the mean square error and the relative assembly costs. The results are listed in Table 5 and depicted graphically in Fig. 1, right. As expected, the behaviour is similar to $d = 3$ but the overall error is smaller. Taking values in the interval

$$\delta \in [0.000977, 0.00391]$$

offers a good compromise between noticeable speedup and error comparable to that of the standard method.

$\hat{m} = \hat{n}$	200	400	800	1600	3200	6400	standard
δ	1	0.25	$6.25e{-2}$	$1.56e{-2}$	$3.91e{-3}$	$9.77e{-4}$	0
MSE/def	$4.7e{-4}$	$2.3e{-4}$	$1.2e{-4}$	$1.2e{-4}$	$3.1e{-5}$	$1.6e{-5}$	$3.0e{-6}$
MSE/pc	$9.8e{-5}$	$4.6e{-5}$	$2.4e{-5}$	$1.2e{-5}$	$6.2e{-6}$	$3.4e{-6}$	$1.2e{-6}$
flops	4.88%	7.36%	12.24%	21.13%	35.27%	53.10%	100%
time	2.59%	3.85%	6.78%	12.75%	20.93%	31.97%	100%

Table 5: Third example – mean square error (MSE) and share of the computational costs for various values of δ and $d = 5$.

4.3. Rate of convergence

We consider again the graph surface of the previous example and use it to investigate the influence of the grid on the rate of convergence in the *fourth example*. More precisely, we randomly sample data from the surface, which is then approximated by dyadically refined bicubic spline surfaces, where the number $m \times n$ of control points varies between 5×5 and 131×131.

The number N of sample points is chosen such that the number of control points per data point evaluates to $\Delta = 0.01$. We consider different grid sizes, which correspond to the four different values

$$\delta_1 = 0.0625,\ \delta_2 = 0.0278,\ \delta_3 = 0.01 \text{ and } \delta_4 = 0.0025$$

of δ. The experimentally observed relative assembly costs are roughly equal to 20%, 25%, 33% and 50%, respectively; we note that in this example the assembly took more than 85% of the total approximation time when using the standard method. The plot in Fig. 3 depicts the impact of the choice of δ on the rate of convergence, which is equal to h^4 when using the standard assembly algorithm, where h denotes the mesh size. We note that the fast assembly method slows the rate of convergence for spline surfaces with many control points. This effect sets in later for smaller values of δ, which, however, increase the relative assembly costs. We conjecture that the rate of convergence tends to the same limit (approximately h^2) for any choice of δ as $h \to 0$.

We show the fitting result of the last configuration ($h = 2^{-7}$) for the standard assembly method and for the fast assembly method with respect to δ_1 in Fig. 2. Since the difference between these results is visible only after significant zooming (cf.

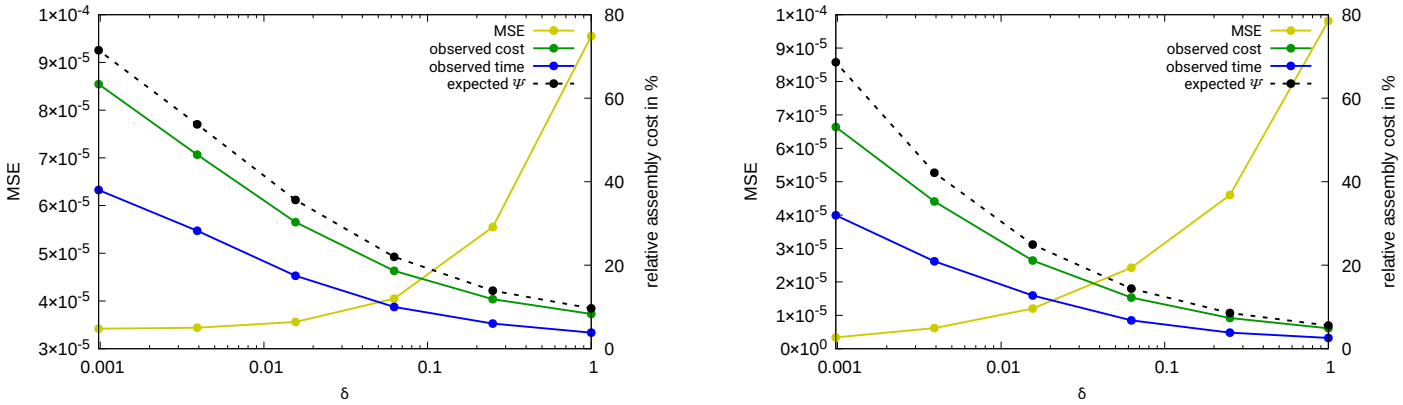


Figure 1: Third example – the trade-off between accuracy and computational costs. Left: $d = 3$, right: $d = 5$

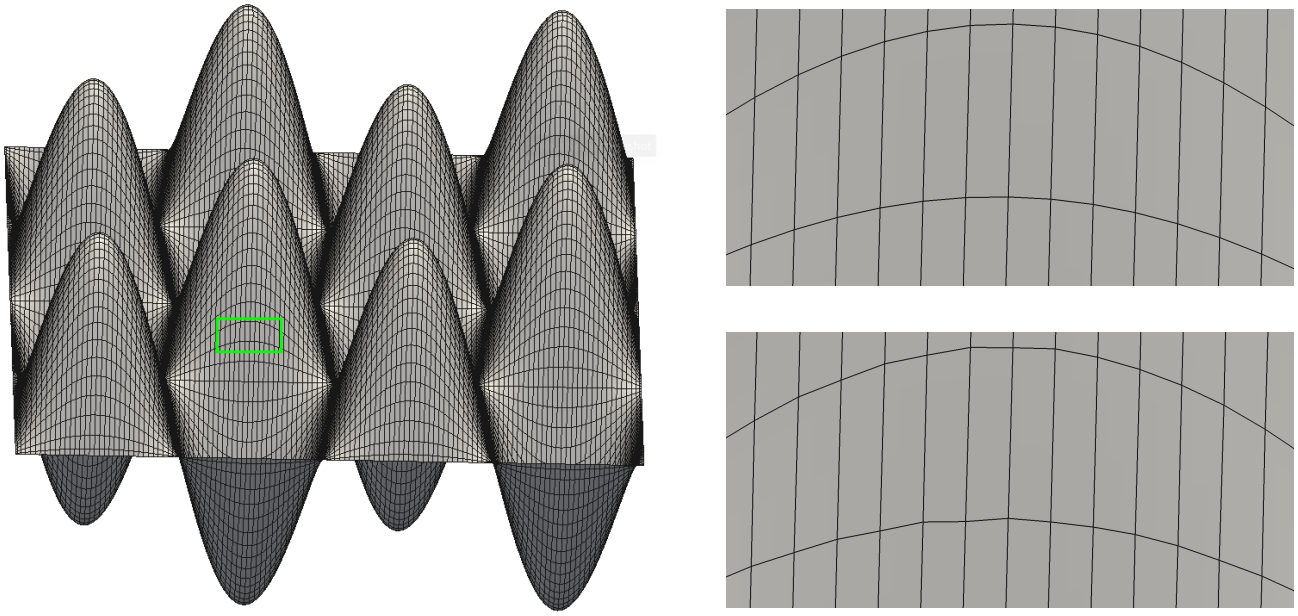


Figure 2: Fourth example – fitting results for $h = 2^{-7}$. Left and bottom right: Fast assembly with δ_1. Top right: Standard assembly.

Fig. 2, right), we show the entire surface only for the fast method in Fig. 2, left. We note that the oscillations of the control net (black lines) induced by the grid projection are barely visible.

4.4. *Influence of noise in the data*

The *fifth example* is devoted to the impact of noise in the data, which is always present in applications. We repeat the computations of the previous example for the largest instance $\delta = 0.0625$ – with relative assembly costs of approximately 20% – but this time we add randomly generated noise to the data. More precisely, the data points are perturbed by random vectors, which are uniformly distributed in balls of radii $r_1 = 2e{-}6$, $r_2 = 1e{-}5$, $r_3 = 5e{-}5$, and $r_4 = 2.5e{-}4$.

Fig. 4 shows the impact of the noise on the mean square error, both for fast (solid lines) and standard (dashed lines) assembly. We observe that the difference of the error is negligible if the noise level is high, while the computational costs of the fast method are significantly lower. For a lower noise level, however, the fast assembly algorithm needs more control points to achieve the same accuracy of the fitting result.

4.5. *Industrial example*

Finally, we present an example from industry, where we approximate a scan of a turbine airfoil consisting of 301,219 points. This is an unusually large number of data for a part of this size. Due to (among other causes) the high computational costs of the standard method, one would normally decimate (and possibly smoothen) the data, in order to arrive at a point cloud with fewer vertices. However, the use of the fast assembly method allows us to deal with the full data set directly.

Fig. 5 visualizes the results obtained by fitting a bicubic spline surface[1] with 50×200 control points and uniform knot vectors to this point cloud, obtained with the standard method (left) and via fast matrix assembly (right) with $\hat{m} = 1000$ and $\hat{n} = 4000$. Note that the surface has a rectangular (i.e., non-square) shape, hence we choose now different values of these two parameters. There are no visible differences between the two surfaces, which indicates that a (well chosen) grid projection has no negative effect on the quality of the resulting surface.

[1] We use a non-periodic spline surface even though the data were scanned from a periodic surface, hence we get a non-watertight result.

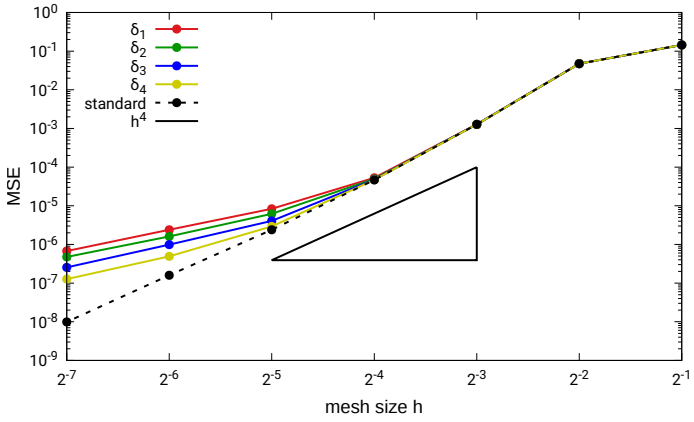


Figure 3: Fourth example – rate of convergence for different values of δ.

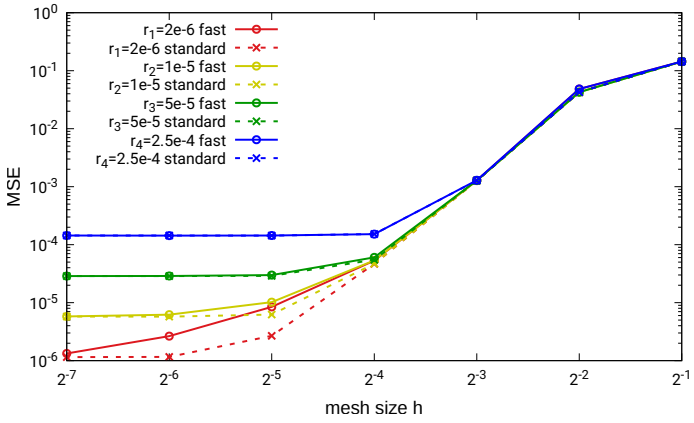


Figure 4: Fifth example – rate of convergence for data with noise.

We investigate the influence of the choice of the grid on the quality and precision of the resulting surface and the computational costs in more detail. Table 6 lists the results for three grid configurations and for the standard method. While the errors are comparable, both in terms of the maximum error and the percentage of points that fulfill the tolerance (10^{-5}m), we note that the computational costs of the fast assembly method are much lower. The experimentally observed numbers of flops are even smaller than the estimate Ψ (note that, unlike in Theorem 3, we have $m \neq n$ and $\hat{m} \neq \hat{n}$). The measured values of the assembly time confirm these observations. Even the finest grid provides a computational advantage, as in this configuration the assembly step takes around 70% of the total fitting time when using the standard method. [2]

Fig. 6 shows the control nets for three grid configurations in a region corresponding to the top left part of Fig. 5, and compares them to the results obtained by the standard method. The reflection lines of the corresponding region are shown for each grid configuration and the standard method in Fig. 7. We can see that the standard method and the finest of the three projection grids (which are the two settings shown in Fig. 5) yield similar mesh quality. While the remaining two configurations give surfaces that satisfy the fitting requirements with respect to the maximum error and the percentage of points fulfilling the tolerance, the oscillating control nets indicate that a wrong choice of $\hat{m}$ and $\hat{n}$ may have a negative impact on the surface quality. This can be observed in the reflection lines as well, where the oscillations from the control net clearly show on the upper edge of the surface in the coarsest setting (Fig. 7 (d)) and are already noticable for the grid size in the middle (Fig. 7 (c)). Note that we did not use any regularization or smoothing terms in this example.

[2] Note however, that for a fixed number of data points h-refinement leads to a bigger linear system while the assembly costs remains more or less the same; consequently, the cost of solving the system dominates as $h \to 0$.

5. Conclusion

Least-squares fitting by tensor-product spline surfaces is a classical approach for approximating unstructured data and is widely used in industry. However, assembling the system of equations via the straightforward approach can be time-consuming. In this paper, we proposed to accelerate this process by employing the technique of sum factorization, which is already a well-established tool in the context of isogeometric analysis. We showed that it is possible to obtain significant gains concerning the computation time. However, we also identified two limitations of the method:

First, the fast assembly method may lead to a reduction of fitting accuracy. In particular, this is true when using coarse grids in the case of highly accurate data. For very precise data, we recommend to employ either the standard assembly algorithm or fine grids, which do not lead to computational disadvantages, as noted in Section 3.3. Nevertheless, the full potential of the method should definitely be exploited in the case of data with a significant level of noise.

Second, the fast assembly algorithm may entail a deterioration of the control mesh quality. Further investigations are needed to understand this phenomenon, and how it can be addressed by invoking additional regularization or fairing terms (which are well known to admit efficient assembly algorithms). However, it is clear that a reasonably fine grid size has to be selected in applications.

Finally, we note that the generalization to adaptive spline refinement, e.g., via hierarchical B-splines, is of vital interest and deserves to be studied in more detail.

Acknowledgement

The authors express their appreciation for supports provided by the Austrian Science Fund (Project No. S11708) and by the European Research Council through the CHANGE project (GA No. 694515). Maodong Pan was also supported by the National Natural Science Foundation of China (No. 12101308) and the Natural Science Foundation of Jiangsu Province, China (No. BK20210268). This support is gratefully acknowledged.

References

[1] J. Hoschek, D. Lasser, Fundamentals of computer aided geometric design, AK Peters, Ltd., 1993.

[2] F. Cheng, A. Goshtasby, A parallel B-spline surface fitting algorithm, ACM Transactions on Graphics 8 (1) (1988) 41–50.

[3] D. Forsey, R. Bartels, Surface fitting with hierarchical splines, ACM Transactions on Graphics (TOG) 14 (2) (1995) 134–161.

[4] V. Weiss, L. Andor, G. Renner, T. Várady, Advanced surface fitting techniques, Computer Aided Geometric Design 19 (1) (2002) 19–42.

[5] V. Krishnamurthy, M. Levoy, Fitting smooth surfaces to dense polygon meshes, in: Proc. 23rd Ann. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH), 1996, pp. 313–324.

[6] M. Eck, H. Hoppe, Automatic reconstruction of B-spline surfaces of arbitrary topological type, in: Proc. 23rd Ann. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH), 1996, pp. 325–334.

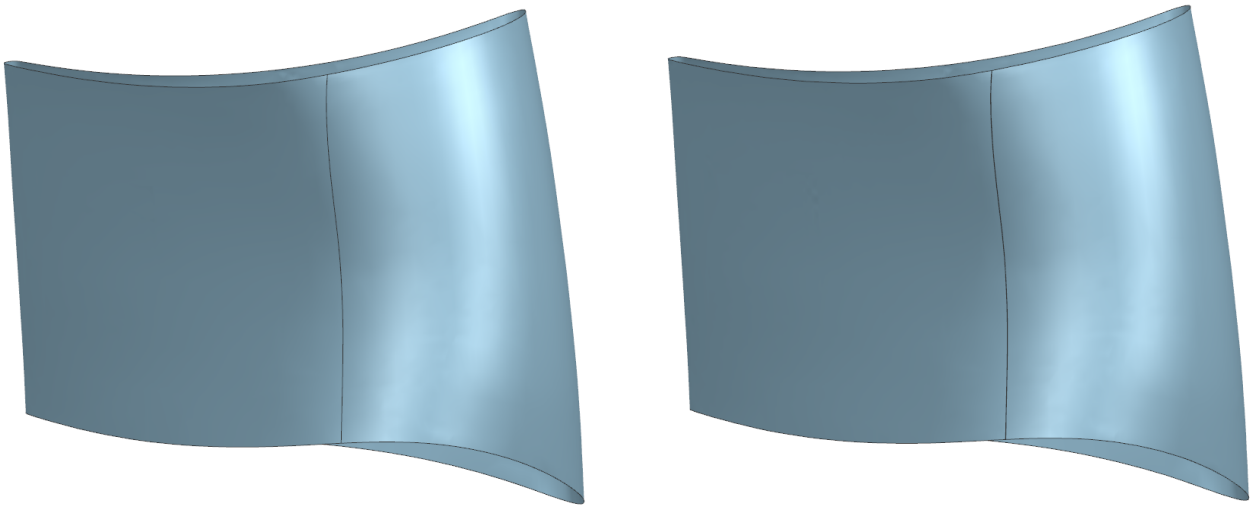


Figure 5: Industrial example – result of least-square fitting with the standard matrix assembly (left) and with the fast matrix assembly.

grid size	maximum error	% of points within tolerance	relative assembly costs		time
			Ψ	experimentally observed	
$\hat{m} = 250$, $\hat{n} = 1000$	$6.84e{-5}$	94.04%	12.47%	11.02%	8.02%
$\hat{m} = 500$, $\hat{n} = 2000$	$6.82e{-5}$	93.95%	19.33%	16.60%	11.86%
$\hat{m} = 1000$, $\hat{n} = 4000$	$6.84e{-5}$	93.85%	31.38%	25.54%	19.35%
standard method, $\hat{m} = \hat{n} = \infty$	$6.84e{-5}$	93.84%	n/a	100%	100%

Table 6: Industrial example – comparison of errors and assembly costs for various grid sizes.

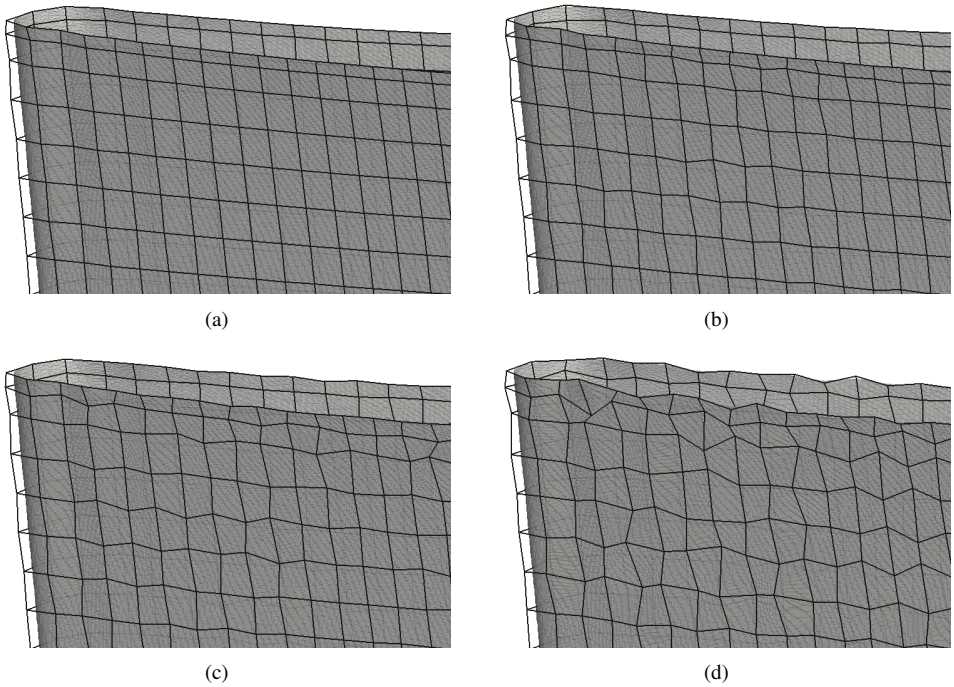


Figure 6: Industrial example – B-spline control nets of the surfaces obtained by least-squares fitting with (a) standard matrix assembly and fast matrix assembly with (b) $\hat{m} = 1000$, $\hat{n} = 4000$, (c) $\hat{m} = 500$, $\hat{n} = 2000$, (d) $\hat{m} = 250$, $\hat{n} = 1000$.

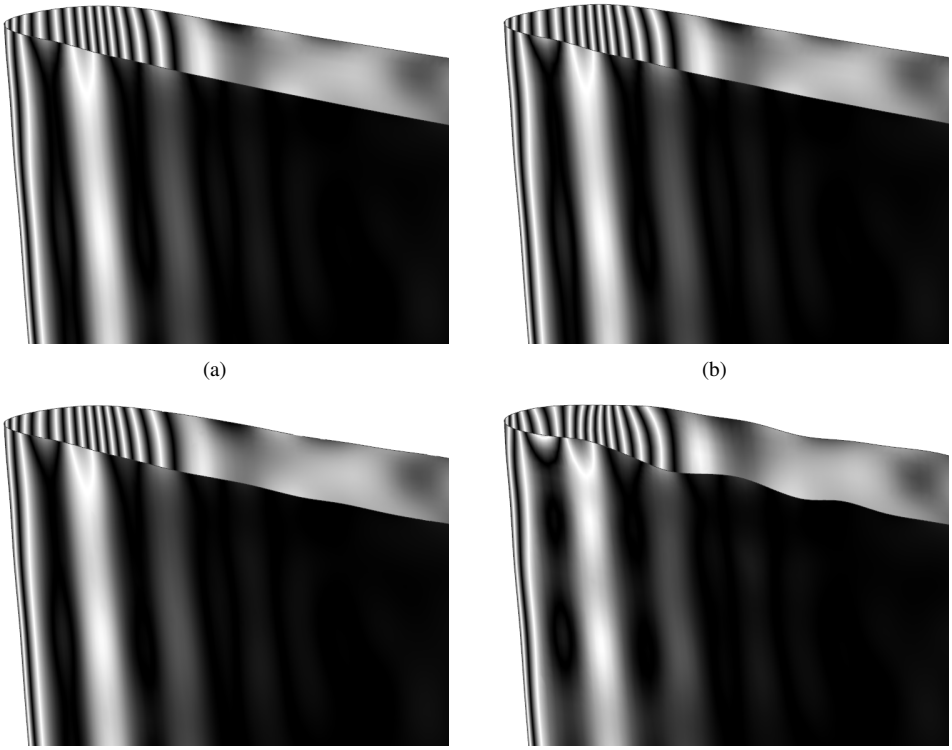
[7] G. Greiner, K. Hormann, Interpolating and approximating scattered 3D data with hierarchical tensor product B-splines, in: Surface Fitting and Multiresolution Methods, Vanderbilt University Press, 1997, pp. 163–172.

[8] T. Várady, R. Martin, Reverse engineering, in: G. Farin, J. Hoschek, M.-S. Kim (Eds.), Handbook of Computer Aided Geometric Design, North-Holland, 2002, Ch. 26, pp. 651–682.

[9] M. Floater, Parametrization and smooth approximation of surface triangulations, Computer Aided Geometric Design 14 (3) (1997) 231–250.

[10] M. Floater, K. Hormann, Surface parameterization: A tutorial and survey, in: Advances in Multiresolution for Geometric Modelling, Springer, Berlin Heidelberg, 2005, pp. 157–186.

[11] J. Hoschek, Intrinsic parametrization for approximation, Computer Aided



(a) (b)

(c) (d)

Figure 7: Industrial example – Enlarged view of the reflection lines of the four surfaces in the previous figure.

Geometric Design 5 (1988) 27–31.

[12] B. Sarkar, C.-H. Menq, Parameter optimization in approximating curves and surfaces to measurement data, Computer Aided Geometric Design 8 (4) (1991) 267–290.

[13] H. Pottmann, S. Leopoldseder, M. Hofer, Approximation with active B-spline curves and surfaces, in: 10th Pacific Conference on Computer Graphics and Applications, IEEE, 2002, pp. 8–25.

[14] B. Jüttler, Surface fitting using convex tensor-product splines, Journal of Computational and Applied Mathematics 84 (1) (1997) 23–44.

[15] J. Hoschek, R. Müller, Turbine blade design by lofted B-spline surfaces, Journal of Computational and Applied Mathematics 119 (1–2) (2000) 235–248.

[16] G. Kiss, C. Giannelli, U. Zore, B. Jüttler, D. Großmann, J. Barner, Adaptive CAD model (re-)construction with THB-splines, Graphical Models 76 (5) (2014) 273–288.

[17] C. Bracco, C. Giannelli, D. Großmann, A. Sestini, Adaptive fitting with THB-splines: Error analysis and industrial applications, Computer Aided Geometric Design 62 (2018) 239–252.

[18] C. Feng, Y. Taguchi, FasTFit: A fast T-spline fitting algorithm, Computer–Aided Design 92 (2017) 11–21.

[19] Z. Lu, X. Jiang, G. Huo, D. Ye, B. Wang, Z. Zheng, A fast T-spline fitting method based on efficient region segmentation, Computational and Applied Mathematics 39 (2020) 55.

[20] G. Kermarrec, P. Morgenstern, Multilevel T-spline approximation for scattered observations with application to land remote sensing, Computer–Aided Design 146 (2022) 103193.

[21] C. Deng, H. Lin, Progressive and iterative approximation for least squares B-spline curve and surface fitting, Computer–Aided Design 47 (2014) 32–44.

[22] M. Liu, B. Li, Q. Guo, C. Zhu, P. Hu, Y. Shao, Progressive iterative approximation for regularized least square bivariate B-spline surface fitting, Journal of Computational and Applied Mathematics 327 (2018) 175–187.

[23] D. Rios, B. Jüttler, LSPIA, (stochastic) gradient descent, and parameter correction, Journal of Computational and Applied Mathematics 406 (2022) 113921.

[24] S. Orszag, Spectral methods for problems in complex geometrics, in: Numerical Methods for Partial Differential Equations, Elsevier, 1979, pp. 273–305.

[25] P. Antolin, A. Buffa, F. Calabrò, M. Martinelli, G. Sangalli, Efficient matrix computation for tensor-product isogeometric analysis: The use of sum factorization, Computer Methods in Applied Mechanics and Engineering 285 (2015) 817–828.

[26] F. Calabrò, G. Sangalli, M. Tani, Fast formation of isogeometric Galerkin matrices by weighted quadrature, Computer Methods in Applied Mechanics and Engineering 316 (2017) 606–622.

[27] A. Bressan, S. Takacs, Sum factorization techniques in isogeometric analysis, Computer Methods in Applied Mechanics and Engineering 352 (2019) 437–460.

[28] R. Hiemstra, G. Sangalli, M. Tani, F. Calabrò, T. Hughes, Fast formation and assembly of finite element matrices with application to isogeometric linear elasticity, Computer Methods in Applied Mechanics and Engineering 355 (2019) 234–260.

[29] M. Pan, B. Jüttler, A. Giust, Fast formation of isogeometric Galerkin matrices via integration by interpolation and look-up, Computer Methods in Applied Mechanics and Engineering 366 (2020) 113005.

[30] M. Pan, B. Jüttler, A. Mantzaflaris, Efficient matrix assembly in isogeometric analysis with hierarchical B-splines, Journal of Computational and Applied Mathematics 390 (2021) 113278.

[31] M. Pan, B. Jüttler, F. Scholz, Efficient matrix computation for isogeometric discretizations with hierarchical B-splines in any dimension, Computer Methods in Applied Mechanics and Engineering 388 (2022) 114210.