Auxiliary Space Preconditioning with a Symmetric Gauss-Seidel Smoothing Scheme for IsoGeometric Discretization of $\mathbf{H_0(curl)}$-elliptic Problem

Abdeladim El Akri, Khalide Jbilou, Nouredine Ouhaddou, and Ahmed Ratnani

1 Introduction

The IsoGeometric Analysis (IgA), introduced by Hughes et al. in [4], is a computational method that provides a general framework for the design and analysis of numerical approximation of partial differential equations (PDEs). The IgA is based on the Galerkin formulation followed by the construction of a finite-dimensional subspace, which approximates the solution space, determined by a finite set of basis functions. These functions are adopted from the geometry description of the PDE domain which usually employs *B*-spline functions, as done by computer-aided design algorithms [6]. As a consequence, the geometry is maintained exactly and the use of high-regularity functions is settled by simply increasing or decreasing the multiplicities of knots.

The discrete problems produced by isogeometric methods are usually very hard to solve by the standard methods; they are ill-conditioned and the development of a preconditioning strategy is not straightforward, specially in the case of problems characterized by the presence of a large kernel of the PDE operator (like e.g the model problem considered in the present paper). In this case a natural way of constructing the preconditioner is the Auxiliary Space Preconditioning (ASP) method introduced by Xu in [9], see also [3]. The latter is a preconditioning technique based on a simple smoothing scheme (e.g Jacobi or Gauss-Seidel method) and an auxiliary space. The method has the main advantage of linking the solution space directly with functions in the potential space, which makes it possible to control the drawback of the presence of a large null space.

Abdeladim El Akri, Nouredine Ouhaddou, Ahmed Ratnani
Lab. MSDA, Mohammed VI Polytechnic University, Green City, Morocco e-mail: abdeladim.elakri@um6p.ma, nouredine.ouhaddou@um6p.ma, ahmed.ratnani@um6p.ma

Khalide Jbilou
Lab. MSDA, Mohammed VI Polytechnic University, Green City, Morocco and Lab. LMPA, University of Littoral Côte d'Opale, Calais cedex, France e-mail: khalide.jbilou@univ-littoral.fr

Due to page limitation, in the present work we consider only one model problem (even if the results are valid for a variety of $\boldsymbol{H}(\mathbf{curl})$ and $\boldsymbol{H}(\mathrm{div})$ problems)

$$\mathbf{curl}\,\mathbf{curl}\,\boldsymbol{u} + \tau \boldsymbol{u} = \boldsymbol{f} \text{ in } \Omega, \quad \boldsymbol{u} \times \boldsymbol{n} = 0 \text{ on } \partial\Omega, \tag{1}$$

where the vector function $\boldsymbol{f} \in (L^2(\Omega))^3$, τ is a positive constant and $\Omega = (0,1)^3$. We develop a fast preconditioned iterative linear solver for (1). The resulting algorithm relies on a symmetric Gauss-Seidel smoothing scheme, Poisson problem solvers, and a GLT-based smoother to remove the dependence on the degree p . For the former we provide a new algorithm which exploits the block representation of the matrix of the resulting discrete system through sum of Kronecker products. For the isogeometric discretization of the Poisson problems, we adopt the fast diagonalization method developed in [8]. The GLT smoother is taken from [5].

The rest of the paper is organized as follows. Section 2 presents the IgA finite element discretization of the model (1). In Section 3, we propose a new algorithm for the symmetric Gauss-Seidel method that utilizes the block structure of the matrix-based discretization of (1). Next, in Section 4, we introduce the auxiliary space preconditioner, and in Section 5, we combine it with a GLT-based smoother to control the p-dependency of the solver. Finally, in Section 6, we illustrate the performance of our preconditioner with several numerical tests.

2 Isogeometric discretization

For the sake of simplicity, we shall consider only *non-periodic* and *uniform* knot vectors of the form

$$T = (\underbrace{0,\ldots,0}_{p+1}, t_{p+2} < t_{p+3} < \ldots t_{n-1} < t_n, \underbrace{1,\ldots,1}_{p+1}),$$

where t_i is the i-th knot, n is the number of basis functions and p is the polynomial order. B-spline basis functions are defined recursively and they begin with order $p = 0$ such as

$$B_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1}, \\ 0 & \text{otherwise} \end{cases}$$

and for higher order $p \geq 1$ as follows

$$B_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} B_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(t),$$

in which a fraction with zero denominator is assumed to be zero. We let

$$\mathcal{S}^p = \operatorname{span}\left\{B_{i,p} \,:\, i = 1,\ldots,n\right\}, \quad \mathcal{S}_0^p = \operatorname{span}\left\{B_{i,p} \,:\, i = 2,\ldots,n-1\right\},$$

the *uni-variate spline* spaces spanned by the B-spline functions. For three-dimensional vector field structures, we specify a tridirectional knot vector $\boldsymbol{T} = T_1 \times T_2 \times T_3$, where each T_i is an open and uniform univariate knot vector related to a B-spline degree p_i. We let then

$$\boldsymbol{V}_{h,0}(\mathbf{curl}) = \left(\mathcal{S}^{p_1-1} \otimes \mathcal{S}_0^{p_2} \otimes \mathcal{S}_0^{p_3}\right) \times \left(\mathcal{S}_0^{p_1} \otimes \mathcal{S}^{p_2-1} \otimes \mathcal{S}_0^{p_3}\right) \times \left(\mathcal{S}_0^{p_1} \otimes \mathcal{S}_0^{p_2} \otimes \mathcal{S}^{p_3-1}\right),$$

the three-dimensional isogeometric approximation of $\boldsymbol{H}_0(\mathbf{curl})$ (see [1]). However, we shall need also the following discrete counterpart of space $H_0^1(\Omega)$

$$V_{h,0}(\mathbf{grad}) = \mathcal{S}_0^{p_1} \otimes \mathcal{S}_0^{p_2} \otimes \mathcal{S}_0^{p_3}.$$

Among the important properties, spaces $V_{h,0}(\mathbf{grad})$ and $\boldsymbol{V}_{h,0}(\mathbf{curl})$ feature quasi interpolation operators $\Pi_{h,0}^{\mathbf{grad}}$ and $\Pi_{h,0}^{\mathbf{curl}}$ (see [1], for instance) that make the (DeRham) diagram

$$\begin{array}{ccc} H_0^1 & \xrightarrow{\mathbf{grad}} & \boldsymbol{H}_0(\mathbf{curl}) \\ \Pi_{h,0}^{\mathbf{grad}} \Big\downarrow & & \Pi_{h,0}^{\mathbf{curl}} \Big\downarrow \\ \boldsymbol{V}_{h,0}(\mathbf{grad}) & \xrightarrow{\mathbf{grad}} & \boldsymbol{V}_{h,0}(\mathbf{curl}) \end{array}$$

commutes and exact.

Our discrete solution $\boldsymbol{u}_h \in \boldsymbol{V}_{h,0}(\mathbf{curl})$ satisfies the weak formulation

$$(\mathbf{curl}\, \boldsymbol{u}_h, \mathbf{curl}\, \boldsymbol{v}_h) + \tau(\boldsymbol{u}_h, \boldsymbol{v}_h) = (\boldsymbol{f}, \boldsymbol{v}_h), \quad \forall \boldsymbol{v}_h \in \boldsymbol{V}_{h,0}(\mathbf{curl}), \tag{2}$$

where $(\cdot,\cdot)$ refers to the $(L^2(\Omega))^3$ inner-product. With the standard basis for $\boldsymbol{V}_{h,0}(\mathbf{curl})$ (see [7]), we can write (2) as a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ is a (symmetric) 3×3 block matrix of the form

$$\boldsymbol{A} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}, \tag{3}$$

where each diagonal block matrix A_{ii} is a sum of Kronecker products of 3 matrices while the non-diagonal matrices A_{ij} $(i \neq j)$ are Kronecker products of 3 matrices. Algorithms presented in the next section exploit this (tensor-product) structure.

3 Block fast Gauss-Seidel method for sum of Kronecker products

In this section we present an efficient implementation of the block Gauss-Seidel method that is specifically designed for solving systems of equations involving a sum of Kronecker product matrices. This implementation is a key contribution of our paper and is used in the Gauss-Seidel smoothing step of the optimal ASP-based algorithm presented in Section 5.

To begin, we recall the symmetric Gauss-Seidel method in Algorithm 1. Our implementation uses the `spsolve` driver, which has different implementations depending on the type of the matrix $\boldsymbol{A}$ (lower or upper triangular matrix). These implementations are given in algorithms 2–3.

Algorithm 1: `Symmetric Gauss Seidel solver`

Input : $\boldsymbol{A}$: A given matrix, $\boldsymbol{b}$: A given vector, $\boldsymbol{x}$: A starting point, ν_1: The number of iterations
Output : $\boldsymbol{x}$: The approximate solution of $\boldsymbol{Ax} = \boldsymbol{b}$

for $i \leftarrow 1$ ***to*** ν_1 **do**
  $\boldsymbol{x} \leftarrow \boldsymbol{x} + \texttt{spsolve}(\boldsymbol{A}, \boldsymbol{b} - \boldsymbol{Ax}, \texttt{lower} = \texttt{True})$
end
for $i \leftarrow 1$ ***to*** ν_1 **do**
  $\boldsymbol{x} \leftarrow \boldsymbol{x} + \texttt{spsolve}(\boldsymbol{A}, \boldsymbol{b} - \boldsymbol{Ax}, \texttt{lower} = \texttt{False})$
end

Algorithm 2: `spsolve`: Lower triangular solver for 3×3 block matrix

Input : $\boldsymbol{A}$: Lower triangular matrix, $\boldsymbol{b}$: A given vector
Output : $\boldsymbol{x}$: Solution of $\boldsymbol{Ax} = \boldsymbol{b}$

$b_1, b_2, b_3 \leftarrow \texttt{unfold}(\boldsymbol{b})$
$x_1 \leftarrow \texttt{spsolve}(A_{11}, b_1, \texttt{lower} = \texttt{True})$
$\tilde{b}_2 \leftarrow b_2 - A_{21}x_1$
$x_2 \leftarrow \texttt{spsolve}(A_{22}, \tilde{b}_2, \texttt{lower} = \texttt{True})$
$\tilde{b}_3 \leftarrow b_3 - A_{31}x_1 - A_{32}x_2$
$x_3 \leftarrow \texttt{spsolve}(A_{33}, \tilde{b}_3, \texttt{lower} = \texttt{True})$
$\boldsymbol{x} \leftarrow \texttt{fold}(x_1, x_2, x_3)$

Algorithm 3: `spsolve`: Upper triangular solver for 3×3 block matrix

Input : $\boldsymbol{A}$: Upper triangular matrix, $\boldsymbol{b}$: A given vector
Output : $\boldsymbol{x}$: Solution of $\boldsymbol{Ax} = \boldsymbol{b}$

$b_1, b_2, b_3 \leftarrow \texttt{unfold}(\boldsymbol{b})$
$x_3 \leftarrow \texttt{spsolve}(A_{33}, b_3, \texttt{lower} = \texttt{False})$
$\tilde{b}_2 \leftarrow b_2 - A_{23}x_3$
$x_2 \leftarrow \texttt{spsolve}(A_{22}, \tilde{b}_2, \texttt{lower} = \texttt{False})$
$\tilde{b}_1 \leftarrow b_1 - A_{12}x_2 - A_{13}x_3$
$x_1 \leftarrow \texttt{spsolve}(A_{11}, \tilde{b}_1, \texttt{lower} = \texttt{False})$
$\boldsymbol{x} \leftarrow \texttt{fold}(x_1, x_2, x_3)$

Next, we provide a new implementation for the lower triangular solver that is used in Algorithm 2 (the upper solver used in Algorithm 3 follows the same rationals). We refer to our driver as spsolve. Since the diagonal block matrices in (3) are sums of Kronecker products of 3 matrices, we can derive efficient matrix-free implementation as described in Algorithm 4 (in the case of a sparse matrix (CSR)).

4 Auxiliary space preconditioner

In this section, we present the auxiliary space preconditioning strategy for system (2). To keep the presentation focused, we only introduce the ASP preconditioner (we refer to [2] for more detailed analysis and further discussion of the preconditioner). For this purpose, we introduce the following matrices

- $\boldsymbol{H}$ defines the matrix related to the restriction of $(H_0^1(\Omega))^3$ inner product to $(V_{h,0}(\mathbf{grad}))^3$, and $\boldsymbol{M}$ is the matrix representation related to the restriction of the $(L^2(\Omega))^3$ inner product to $(V_{h,0}(\mathbf{grad}, \Omega))^3$.

Algorithm 4: `spsolve`: Lower triangular solver for sum of Kronecker product [CSR] matrices.

Input : $\boldsymbol{A}$: Lower triangular matrix of the form $\alpha A_1 \otimes A_2 \otimes A_3 + \beta B_1 \otimes B_2 \otimes B_3 + \gamma C_1 \otimes C_2 \otimes C_3$, $\boldsymbol{b}$: A given vector

Output : $\boldsymbol{x}$: Solution of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$

```
// n_l is the number of rows of matrices A_l, B_l, C_l, l = 1,2,3.
for i_1 ← 1 to n_1 do
    for i_2 ← 1 to n_2 do
        for i_3 ← 1 to n_3 do
            i ← multi_index(i_1, i_2, i_3)
            y_i ← 0
            a_d ← 1
            for k_1 ← A_1.indptr[i_1] to A_1.indptr[i_1 + 1] − 1 do
                j_1 ← A_1.indices[k_1]
                a_1 ← A_1.data[k_1]
                for k_2 ← A_2.indptr[i_2] to A_2.indptr[i_2 + 1] − 1 do
                    j_2 ← A_2.indices[k_2]
                    a_2 ← A_2.data[k_2]
                    for k_3 ← A_3.indptr[i_3] to A_3.indptr[i_3 + 1] − 1 do
                        j_3 ← A_3.indices[k_3]
                        a_3 ← A_3.data[k_3]
                        j ← multi_index(j_1, j_2, j_3)
                        if i < j then
                            y_i ← y_i + a_1 a_2 a_3 x[j]
                        else
                            a_d ← a_1 a_2 a_3
                        end
                    end
                end
            end
            z_i ← 0
            b_d ← 1
            for k_1 ← B_1.indptr[i_1] to B_1.indptr[i_1 + 1] − 1 do
                j_1 ← B_1.indices[k_1]
                a_1 ← B_1.data[k_1]
                for k_2 ← B_2.indptr[i_2] to B_2.indptr[i_2 + 1] − 1 do
                    j_2 ← B_2.indices[k_2]
                    a_2 ← B_2.data[k_2]
                    for k_3 ← B_3.indptr[i_3] to B_3.indptr[i_3 + 1] − 1 do
                        j_3 ← B_3.indices[k_3]
                        a_3 ← B_3.data[k_3]
                        j ← multi_index(j_1, j_2, j_3)
                        if i < j then
                            z_i ← z_i + a_1 a_2 a_3 x[j]
                        else
                            b_d ← a_1 a_2 a_3
                        end
                    end
                end
            end
            w_i ← 0
            c_d ← 1
            for k_1 ← C_1.indptr[i_1] to C_1.indptr[i_1 + 1] − 1 do
                j_1 ← C_1.indices[k_1]
                a_1 ← C_1.data[k_1]
                for k_2 ← C_2.indptr[i_2] to C_2.indptr[i_2 + 1] − 1 do
                    j_2 ← C_2.indices[k_2]
                    a_2 ← C_2.data[k_2]
                    for k_3 ← C_3.indptr[i_3] to C_3.indptr[i_3 + 1] − 1 do
                        j_3 ← C_3.indices[k_3]
                        a_3 ← C_3.data[k_3]
                        j ← multi_index(j_1, j_2, j_3)
                        if i < j then
                            w_i ← w_i + a_1 a_2 a_3 x[j]
                        else
                            c_d ← a_1 a_2 a_3
                        end
                    end
                end
            end
            x[i] ← 1/(α a_d + β b_d + γ c_d) (b[i] − α y_i − β z_i − γ w_i)
        end
    end
end
```

- We write $\boldsymbol{P}$ and $\boldsymbol{G}$ for matrices related to the transform operators $\Pi_{h,0}^{\mathbf{curl}}|_{(V_{h,0}(\mathbf{grad}))^3}$ and $\mathbf{grad}|_{V_{h,0}(\mathbf{grad})}$, respectively.
- Let $\boldsymbol{L}$ be the matrix related to the mapping

$$(\boldsymbol{\phi}_h, \widetilde{\boldsymbol{\phi}_h}) \in V_{h,0}(\mathbf{grad}) \times V_{h,0}(\mathbf{grad}) \longmapsto \left(\mathbf{grad}\, \boldsymbol{\phi}_h, \mathbf{grad}\, \widetilde{\boldsymbol{\phi}_h}\right).$$

- $\boldsymbol{S}$ stands for the matrix related to the smoother.

With these notations, ASP preconditioner for problem (2) is given by

$$\boldsymbol{B} = \boldsymbol{S} + \boldsymbol{K}, \quad \boldsymbol{K} := \boldsymbol{P}\,(\boldsymbol{H} + \tau \boldsymbol{M})^{-1}\,\boldsymbol{P}^T + \tau^{-1}\boldsymbol{G}\boldsymbol{L}^{-1}\boldsymbol{G}^T. \tag{4}$$

The smoother $\boldsymbol{S}$ can be chosen by a simple relaxation scheme such as the Jacobi and symmetric Gauss-Seidel (GS) method. In this case, it has been proved in [2] that the spectral condition number $\kappa(\boldsymbol{BA})$ is bounded, with respect to discretization parameter h. However, the numerical tests developed in the aforementioned paper show that the overall performance obtained with Gauss-Seidel smoother is better than that obtained with Jacobi. That's why in the present paper we focus on the symmetric Gauss-Seidel method.

5 hp-Robust preconditioning algorithm

In this section, we introduce the ASP-GS-GLT algorithm, which is based on the ASP method and addresses the problem related to the B-Spline degree. Indeed, the ASP approach can be extended to construct a p-stable preconditioner by incorporating an extra smoother that controls the p-dependency of the preconditioner. To derive the smoother, we use the theory of Generalized Locally Toeplitz (GLT) sequences (see [5]).

The ASP-GS-GLT algorithm is formulated using the decomposition (4) as follows:

Algorithm 5: ASP-GS-GLT preconditioning for $\boldsymbol{V}_{h,0}(\mathbf{curl})$

Input : $\boldsymbol{A}$: The matrix given in (3), $\boldsymbol{b}$: A given vector, $\boldsymbol{x}$: A starting point, ν_1: The number of GS iterations , ν_2: The number of GLT iterations , ν_{ASP}: The number of ASP iterations
Output : $\boldsymbol{x}$: The approximate solution of $\boldsymbol{Ax} = \boldsymbol{b}$

$k \leftarrow 0$
while $k \leq \nu_{ASP}$ ***and*** *not convergence* **do**
  $\boldsymbol{x} \leftarrow \mathtt{smoother}_1(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{x}, \nu_1)$ `// Apply Symmetric GS smoother`
  $\boldsymbol{x} \leftarrow \mathtt{smoother}_2(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{x}, \nu_2)$ `// Apply GLT smoother`
  $\boldsymbol{d} \leftarrow \boldsymbol{b} - \boldsymbol{Ax}$ `// Compute the defect`
  $\boldsymbol{x}_c \leftarrow \boldsymbol{K}\,\boldsymbol{d}$ `// ASP correction`
  $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{x}_c$ `// Update the solution`
  $k \leftarrow k + 1$
end

Algorithm 5 is built upon three building blocks: a symmetric Gauss-Seidel smoothing, a GLT-based smoother, and an ASP correction. To implement the Gauss-Seidel smoothing, we employ the block fast Gauss-Seidel method described in Section 3.

Our GLT-smoothing strategy is adapted from the work of [5]. Additionally, the ASP correction utilizes solvers for Poisson problems to compute solutions for systems with matrices $\boldsymbol{H} + \tau \boldsymbol{M}$ and $\boldsymbol{L}$. For this purpose, we rely on the fast diagonalization method introduced in [8].

6 Numerical results

In this section, we present some numerical experiments to test the strategy proposed in this paper in view of further applications. In all these tests, we consider the model problem (1) in the computational domain $\Omega = (0,1)^3$ subdivided into $2^k \times 2^k \times 2^k$ sub-domains ($k \geq 1$). As a right-hand side function we chose $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, z) = (\boldsymbol{x}, \boldsymbol{y}, z)$. The IgA discrete system (2) is solved by the Conjugate Gradient (CG) method in the case of the un-preconditioned and preconditioned systems. The stopping criteria is $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|/\|\boldsymbol{b}\| \leq 10^{-6}$ and the initial guess is chosen to be the zero vector.

Table 1 Un-preconditioned (NP) and ASP preconditioner (ASP): CG iterations counts for different values of $h = 1/2^k$ and p. '−' means that CG reaches the maximum number of iterations (set to 3000) without convergence. Parameter values $\tau = 10^{-4}$, $\nu_1 = 1$, $\nu_2 = p + 1$ and $\nu_{asp} = 3$.

	$h = 1/8$		$h = 1/16$		$h = 1/32$		$h = 1/64$			$h = 1/8$		$h = 1/16$		$h = 1/32$		$h = 1/64$	
p	NP	ASP	NP	ASP	NP	ASP	NP	ASP	p	NP	ASP	NP	ASP	NP	ASP	NP	ASP
1	151	3	328	4	511	6	879	6	6	–	4	–	4	–	4	–	4
2	520	2	975	4	1313	5	1962	6	7	–	4	–	4	–	4	–	4
3	–	2	–	3	–	4	–	6	8	–	4	–	4	–	4	–	4
4	–	3	–	3	–	4	–	5	9	–	5	–	4	–	4	–	5
5	–	3	–	3	–	4	–	5	10	–	5	–	5	–	5	–	5

Table 2 ASP preconditioner: CG iterations counts for different values of τ and p. '−' means that CG reaches the maximum number of iterations (set to 3000) without convergence. Parameter values $h = 1/64$, $\nu_1 = 1$, $\nu_2 = p + 1$ and $\nu_{asp} = 3$.

	$p = 1$		$p = 3$		$p = 8$			$p = 1$		$p = 3$		$p = 8$	
τ	NP	ASP	NP	ASP	NP	ASP	τ	NP	ASP	NP	ASP	NP	ASP
10^{-4}	879	6	–	6	–	4	10	244	6	1180	5	–	4
10^{-3}	755	6	–	6	–	4	10^2	131	6	361	4	2227	3
10^{-2}	610	7	–	6	–	4	10^3	41	4	101	2	687	4
10^{-1}	486	7	–	5	–	4	10^4	10	1	39	2	320	3
1	295	7	2278	5	–	4	10^5	9	1	39	1	318	2

In the first test, we keep following the number of the CG iterations for convergence for different values of k and p. The results are shown in Table 1. As we can observe from the table, this example indicates that our ASP preconditioner is robust in the sense that the number of iterations necessary to achieve the convergence is sufficiently small and is hardly dependent on the mesh parameter h and the B-spline degree p.

In the second test, we study the dependence of the ASP preconditioner on the parameter τ. For this objective, in Table 2 we provide GC iteration counts for different values of τ and p. The table shows a strong dependence of the un-preconditioned problem on τ, In contrast, however, the number of CG iterations, in the case of ASP preconditioner, is independent of τ. This shows that the ASP method is perfectly able to handle small values of τ.

Acknowledgements This action benefited from the support of the Chair "Multiphysics and HPC" led by Mohammed VI Polytechnic University, sponsored by OCP.

References

1. Da Veiga, L. B., Buffa, A., Sangalli, G., and Vázquez, R. Mathematical analysis of variational isogeometric methods. *Acta Numerica* **23**, 157–287 (2014).
2. El Akri, A., Jbilou, K., and Ratnani, A. Auxiliary splines space preconditioning for b-spline finite elements: the case of $\mathbf{H}(\mathbf{curl}, \omega)$ and $\mathbf{H}(\mathrm{div}, \omega)$ elliptic probelems. *arXiv preprint arXiv:2303.08375* (2023).
3. Hiptmair, R. and Xu, J. Nodal auxiliary space preconditioning in $\mathbf{H}(\mathbf{curl}, \omega)$ and $\mathbf{H}(\mathrm{div}, \omega)$ spaces. *SIAM Journal on Numerical Analysis* **45**(6), 2483–2509 (2007).
4. Hughes, T., Cottrell, J., and Bazilevs, Y. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering* **194**(39-41), 4135–4195 (2005).
5. Mazza, M., Manni, C., Ratnani, A., Serra-Capizzano, S., and Speleers, H. Isogeometric analysis for 2d and 3d $\mathbf{curl}-\mathrm{div}$ problems: Spectral symbols and fast iterative solvers. *Computer Methods in Applied Mechanics and Engineering* **344**, 970–997 (2019).
6. Piegl, L. and Tiller, W. *The NURBS book*. Springer Science & Business Media (1996).
7. Ratnani, A. and Sonnendrücker, E. An arbitrary high-order spline finite element solver for the time domain maxwell equations. *Journal of Scientific Computing* **51**(1), 87–106 (2012).
8. Sangalli, G. and Tani, M. Isogeometric preconditioners based on fast solvers for the sylvester equation. *SIAM Journal on Scientific Computing* **38**(6), A3644–A3671 (2016).
9. Xu, J. The auxiliary space method and optimal multigrid preconditioning techniques for unstructured grids. *Computing* **56**(3), 215–235 (1996).