HAZniCS – Software Components for Multiphysics Problems

ANA BUDIŠA, Simula Research Laboratory, Norway
XIAOZHE HU, Department of Mathematics, Tufts University, USA
MIROSLAV KUCHTA, Simula Research Laboratory, Norway
KENT–ANDRÉ MARDAL, Department of Mathematics, University of Oslo, Norway
LUDMIL T. ZIKATANOV, Department of Mathematics, Penn State, USA

We introduce the software toolbox HAZniCS for solving interface-coupled multiphysics problems. HAZniCS is a suite of modules that combines the well-known FEniCS framework for finite element discretization with solver and graph library HAZmath. The focus of the paper is on the design and implementation of robust and efficient solver algorithms which tackle issues related to the complex interfacial coupling of the physical problems often encountered in applications in brain biomechanics. The robustness and efficiency of the numerical algorithms and methods is shown in several numerical examples, namely the Darcy-Stokes equations that model flow of cerebrospinal fluid in the human brain and the mixed-dimensional model of electrodiffusion in the brain tissue.

CCS Concepts: • **Mathematics of computing** → **Solvers**; • **Computing methodologies** → *Modeling and simulation.*

Additional Key Words and Phrases: Mathematical Software, Partial Differential Equations, Multiphysics, Preconditioning, FEniCS Project, HAZmath

1 INTRODUCTION

The present paper introduces a novel collection of tools for interface-coupled multiphysics problems modeled by partial differential equations (PDEs). We target applications where the interface is a main driver of the processes, and where solution strategies relying on decoupled single-physics problems suffer from slow convergence. Our focus is on the multiphysics problems with geometrically complex interfaces and slow dynamics, naturally promoting monolithic solvers. Specifically, we exploit fractional operators and low-order interface perturbations as preconditioning techniques.

The motivation for fractional interface operators comes from the fact that the traces of H^1 and $H(\mathrm{div})$ functions are usually sought in $H^{1/2}$ and $H^{-1/2}$, respectively. Such reasoning is exploited, for instance, in domain decomposition algorithms using Poincaré-Steklov operators [Agoshkov 1988; Badia et al. 2009; Deparis et al. 2006; Quarteroni and Valli 1991], as these operators are spectrally equivalent to a fractional Laplacian operator. Robustness with respect to all material parameters has, however, been hard to obtain. Fractional and interface operators have been exploited for monolithic schemes targeting stability and error estimates [Ambartsumyan et al. 2018; Caucao et al. 2022; Galvis and Sarkis 2007; Layton et al. 2002; Rivière and Yotov 2005], but consequences

Authors' addresses: Ana Budiša, ana@simula.no, Simula Research Laboratory, Kristian Augusts gate 23, 0164, Oslo, Norway; Xiaozhe Hu, Department of Mathematics, Tufts University, 503 Boston Avenue, Medford, 02155, Massachusetts, USA, xiaozhe.hu@tufts.edu; Miroslav Kuchta, Simula Research Laboratory, Kristian Augusts gate 23, 0164, Oslo, Norway, miroslav@simula.no; Kent–André Mardal, Department of Mathematics, University of Oslo, P.O. Box 1053, Blindern, 0316, Oslo, Norway, kent-and@math.uio.no; Ludmil T. Zikatanov, Department of Mathematics, Penn State, 239 McAllister Building, University Park, 16802, Pennsylvania, USA, ludmil@psu.edu.

0098-3500/2023/10-ART $15.00
https://doi.org/10.1145/3625561

for preconditioning have only recently been addressed. Specifically, fractional and metric interface operators (and sums thereof) have uncovered the structure of parameter robust preconditioners for monolithic schemes, in particular for Darcy-Stokes problems [Boon et al. 2022a,b; Holter et al. 2020, 2021; Kuchta et al. 2021], but have been challenging to construct and implement in an efficient fashion. Our focus is to present algorithms for fractional and metric operators that can be conveniently implemented.

From the numerical side, fast solution algorithms for single fractional Laplacians have been proposed based on multilevel approaches [Bærland 2019; Bærland et al. 2019; Bramble et al. 2000; Führer 2022; Zhao et al. 2017] and rational approximations [Harizanov et al. 2020, 2018, 2022]. Here, we explore the latter for the sums of fractional Laplacians required for certain Darcy-Stokes type of problems, based on our recent promising results [Budiša et al. 2023b]. Additionally, as seen in $3d$-$1d$ problems [D'Angelo and Quarteroni 2008; Laurino and Zunino 2019], Biot-Stokes [Boon et al. 2022a; Li and Yotov 2022], and EMI equations [Tveito et al. 2017], lower-order perturbations arise because interface conditions, such as the balance of forces, are often expressed in terms of differences of a quantity across the interface rather than the quantity itself. Therefore, we also consider multilevel algorithms that work robustly in the presence of strong perturbations, i.e. metric terms, at interfaces. That is, we implement multilevel algorithms with a space decomposition aware of the metric kernel [Budiša et al. 2023a].

The software tools we develop aim to solve computational mesoscale multiphysics problems. By computational mesoscale, in this context, we refer to problems in the range of a few hundred thousand to tens of millions of degrees of freedom. These problems may benefit significantly from advanced algorithms, whether or not parallel computing is used. The collection of tools presented in this paper are FEniCS [Logg et al. 2012] add-ons for block assembly [Kuchta 2021] and block preconditioning [Mardal and Haga 2012] combined with a flexible algebraic multigrid (AMG) toolbox, implemented in C, called HAZmath [Adler et al. 2009]. Hence, we have named the tool collection HAZniCS. One of the reasons for developing HAZniCS is precisely the mentioned flexibility and variety of the implementation of the AMG method in HAZmath. It allows us to easily modify available linear solvers and preconditioners or create new model-specific solvers for the multiphysics problems at hand. Additionally, with HAZniCS, we provide another wide range of efficient computational methods for solving PDEs with FEniCS, but also a bridge to Python for HAZmath to be used with other PDE simulation tools. Further in the paper, we highlight with a series of code snippets the implementation of several solvers, namely the aggregation-based and metric-perturbed AMG methods and the rational approximation method.

Moreover, we consider a series of examples of multiphysics problems mainly related to biomechanical applications. Namely, we include: (1) a simple three-dimensional example of an elliptic problem on a regular domain, (2) Darcy-Stokes equations describing the interaction of the viscous flow of cerebrospinal fluid surrounding the brain and interacting with the porous media flow of interstitial fluid inside the brain, and (3) the mixed-dimensional equations representing electric signal propagation in neurons and the surrounding matter.

The outline of the current paper is as follows: in Section 2 we introduce the multiphysics models together with the necessary mathematical concepts and numerical methods. Section 3 focuses on the implementation of those methods and the interface between the software components. In Section 4 we present the solver capabilities of our software to simulate relevant biomechanical phenomena. Finally, we draw concluding remarks in Section 5.

2 EXAMPLES

The following three examples introduce different single- and multiphysics PDE models, as well as the relevant mathematical and computational concepts. More specifically, the examples provide an overview of iterative methods and preconditioning techniques for interface-coupled problems that lead, e.g., to the utilization of sums of fractional operators weighted by material parameters.

2.1 Linear elliptic problem

We start with a linear elliptic problem on a three-dimensional ($3d$) regular domain. This example will serve as a baseline for the solvers in HAZniCS. Our goal is to demonstrate that our solver performance is comparable to other established software and to introduce the notation required to form rational approximations of (sums of) fractional operators on the matrix level.

Let $\Omega = [0,1]^3$ be the unit cube and let $\partial\Omega$ denote its boundary. Given external force $f : \Omega \to \mathbb{R}$ and the boundary data $g : \partial\Omega \to \mathbb{R}$, we set to find the solution $u : \Omega \to \mathbb{R}$ that satisfies

$$-\Delta u + u = f \qquad \text{in } \Omega, \tag{1a}$$

$$\frac{\partial u}{\partial \boldsymbol{n}} = g \qquad \text{on } \partial\Omega. \tag{1b}$$

Further, let $L^2 = L^2(\Omega)$ be the space of square-integrable functions on Ω and $H^s = H^s(\Omega)$ the Sobolev spaces with s derivatives in L^2. The corresponding inner products and norms for any function space X are denoted with $(\cdot,\cdot)_X$ and $\|\cdot\|_X$, respectively. Now, let $V_h \subset H^1(\Omega)$ be a finite element space on triangulation of Ω, e.g., of continuous piecewise linear functions ($\mathbb{P}_1$). A discrete variational formulation of (1) is: Find $u \in V_h$ such that

$$a(u,v) = \ell(v) \qquad \forall v \in V_h, \tag{2}$$

with $a(u,v) = (u,v)_{L^2(\Omega)} + (\nabla u, \nabla v)_{L^2(\Omega)}$ and $\ell(v) = (f,v)_{L^2(\Omega)} + (g,v)_{L^2(\partial\Omega)}$.

The corresponding system of linear equations reads: Find $\mathsf{u} \in \mathbb{R}^m$ such that

$$\mathsf{A}\mathsf{u} = \mathsf{b}, \tag{3}$$

with m the dimension of finite element space V_h, $\mathsf{A} \in \mathbb{R}^{m\times m}$ the system matrix representing bilinear form $a(\cdot,\cdot)$ and vector $\mathsf{b} \in \mathbb{R}^m$ representing the right-hand side functional $\ell(\cdot)$. We remark that u and b above are both vectors in $\mathbb{R}^m$, but u is in the so-called *nodal* representation, while b is in the *dual* representation [Bramble 2019; Mardal and Winther 2011]. As such, the matrix A maps the nodal representations of $\mathbb{R}^m$ to their dual representations, while the inverse of the matrix or a *preconditioner* is a mapping from the dual to the nodal representation.

In our case, it is well-known that multilevel methods, such as AMG, provide spectrally equivalent and order optimal algorithms for the inverses of discretizations of the operator $I - \Delta$. We describe the AMG method and its implementation in more detail in Section 3.1 and showcase the performance of HAZmath AMG compared with HYPRE AMG [Falgout and Yang 2002] in Section 4.1.

2.2 Modeling brain clearance during sleep with Darcy-Stokes equations

We consider a multiphysics problem arising in modeling processes of waste clearance in the brain during sleep [Eide et al. 2021; Xie et al. 2013] with potential links to the development of Alzheimer's disease. A novel model, called the glymphatic system [Iliff et al. 2012], states that the viscous flow of cerebrospinal fluid (CSF) is tightly coupled to the porous flow in the brain tissue and that during sleep, in particular, it clears metabolic waste from the brain. For a recent review see [Kelley et al. 2022] and for computational models similar to the one presented here see [Boon et al. 2022a; Holter et al. 2020; Hornkjøl et al. 2022; Kedarasetti et al. 2020]. To this end, we consider patient-specific geometries generated from MR images by the SVTMK library [Mardal et al. 2022] used in [Boon et al. 2022a], see Figure 1. Using SVMTK, the segmented brain geometry is enclosed in a thin shell which, together with the ventricles (the orange subregion in Figure 1), makes up the Stokes domain. The diameter of the Stokes domain is roughly 15 cm while the shell thickness is on average 0.8 mm.

In order to model the waste clearance, let $\Omega_D \subset \mathbb{R}^d$, $d = 2, 3$ be the domain of the porous medium flow that represents the brain tissue[1], and let $\Omega_S \subset \mathbb{R}^d$ be the domain of viscous flow representing the subarachnoid space

[1]In the context of brain mechanics, the case $d = 2$ is relevant, e.g., for slices of the brain geometry.

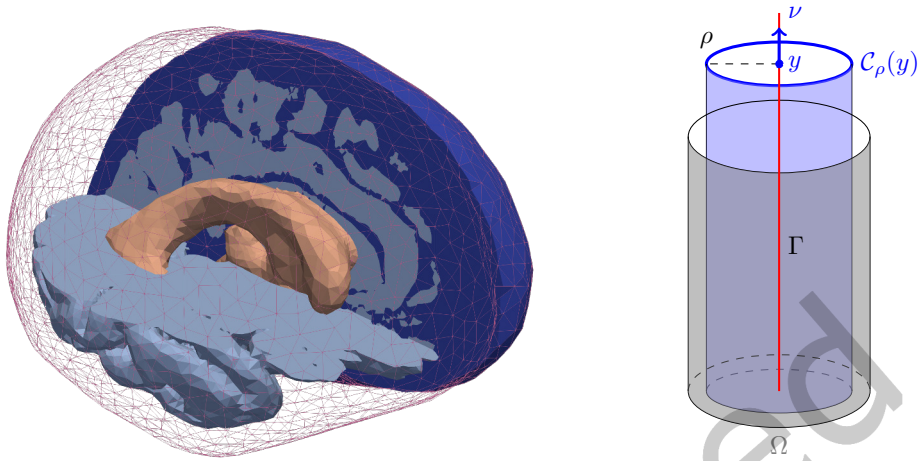


Fig. 1. (Left) Geometry and computational mesh from [Boon et al. 2022a] of the Darcy-Stokes model of brain clearance. Mesh and indicator functions for tracking subdomains making up the Darcy (light blue) and the Stokes domains (dark blue and orange subregions) and their interfaces are generated with SVMTK [Mardal et al. 2022]. (Right) Schematic model reduction from 3d-3d to 3d-1d problem. Idealized dendrites (in blue) are reduced to their centerline while the coupling with the surrounding Ω is accounted for by averaging over-idealized cylindrical surfaces with radius ρ.

around it saturated by CSF. Let Γ denote the interface between the domains, which in this case corresponds to the surface of the brain. We then consider the Darcy-Stokes model which seeks to find Stokes velocity $\boldsymbol{u}_S : \Omega_S \to \mathbb{R}^d$ and pressure $p_S : \Omega_S \to \mathbb{R}$, and Darcy velocity $\boldsymbol{u}_D : \Omega_D \to \mathbb{R}^d$ and pressure $p_D : \Omega_D \to \mathbb{R}$ that satisfy

$$-\nabla \cdot \boldsymbol{\sigma}_S(\boldsymbol{u}_S, p_S) = \boldsymbol{f}_S \quad \text{and} \quad \nabla \cdot \boldsymbol{u}_S = 0 \quad \text{in } \Omega_S, \tag{4a}$$

$$\boldsymbol{u}_D + K\nabla p_D = 0 \quad \text{and} \quad \nabla \cdot \boldsymbol{u}_D = f_D \quad \text{in } \Omega_D, \tag{4b}$$

with interface conditions

$$\boldsymbol{u}_S \cdot \boldsymbol{n} - \boldsymbol{u}_D \cdot \boldsymbol{n} = 0 \quad \text{on } \Gamma, \tag{4c}$$

$$\boldsymbol{n} \cdot \boldsymbol{\sigma}_s(\boldsymbol{u}_S, p_S) \cdot \boldsymbol{n} + p_D = 0 \quad \text{on } \Gamma, \tag{4d}$$

$$\boldsymbol{n} \cdot \boldsymbol{\sigma}_s(\boldsymbol{u}_S, p_S) \cdot \boldsymbol{\tau} + D\boldsymbol{u}_S \cdot \boldsymbol{\tau} = 0 \quad \text{on } \Gamma. \tag{4e}$$

Here, $\boldsymbol{\sigma}_S(\boldsymbol{u}_S, p_S) = \mu\nabla\boldsymbol{u}_S - p_S I$. We remark that for simplicity, $\boldsymbol{\sigma}_S$ is defined in terms of the full velocity gradient and not only its symmetric part, cf. [Layton et al. 2002]. The parameters μ, K, and D are positive constants related to the problem's physical parameters, i.e., the fluid viscosity, permeability, and the Beavers-Joseph-Saffman (BJS) coefficient. Functions $\boldsymbol{f}_S$ and f_D represent the external forces. Additionally, $\boldsymbol{n}$ denotes the unit outer normal of Ω_S and $\boldsymbol{\tau}$ is any unit vector tangent to the interface. In particular, for $d = 3$ the condition (4e) represents a pair of constraints. Finally, we assume the following boundary conditions, for $\partial\Omega_{S,D} \cup \partial\Omega_{S,N} = \partial\Omega_S \backslash \Gamma$, $\partial\Omega_{S,D} \cap \partial\Omega_{S,N} = \emptyset$,

$$\boldsymbol{u}_S = \boldsymbol{0} \quad \text{on } \partial\Omega_{S,D}, \tag{5a}$$

$$\boldsymbol{\sigma}_S(\boldsymbol{u}_S, p_S) \cdot \boldsymbol{n} = \boldsymbol{g} \quad \text{on } \partial\Omega_{S,N}. \tag{5b}$$

To arrive at the finite element formulation of (4) we introduce a Lagrange multiplier $\lambda : \Gamma \to \mathbb{R}$, $\lambda \in \Lambda = \Lambda(\Gamma)$ for enforcing the mass conservation across the interface (4c). In addition, we consider conforming discrete subspaces $\boldsymbol{V}_S \times Q_S \subset \boldsymbol{H}^1(\Omega_S) \times L^2(\Omega_S)$ and $\boldsymbol{V}_d \times Q_D \subset \boldsymbol{H}(\mathrm{div}, \Omega_D) \times L^2(\Omega_D)$ for the Stokes and Darcy subproblems, respectively. In the numerical examples, such spaces are constructed by Taylor-Hood ($\mathbb{P}_2$-$\mathbb{P}_1$) elements and lowest

order Raviart-Thomas elements $\mathbb{RT}_0$ paired with piecewise constant discontinuous Lagrange elements $\mathbb{P}_0^{\text{disc}}$ for Q_D. The multiplier space is discretized by $\mathbb{P}_0^{\text{disc}}$ elements. Let operators $T_{\boldsymbol{n}}$ and $T_{\boldsymbol{\tau}}$ denote the normal and the tangential trace operators on Γ. Then, the discrete system corresponding to (4) states to find $(\boldsymbol{u}_S, p_S, \boldsymbol{u}_D, p_D, \lambda) \in \boldsymbol{V}_s \times Q_s \times \boldsymbol{V}_d \times Q_d \times \Lambda$ that satisfy

$$\underbrace{\begin{pmatrix} -\mu\nabla\cdot\nabla + DT_{\boldsymbol{\tau}}'T_{\boldsymbol{\tau}} & -\nabla & & T_{\boldsymbol{n}}' \\ \nabla\cdot & & & \\ & K^{-1}I & -\nabla & -T_{\boldsymbol{n}}' \\ & \nabla\cdot & & \\ T_{\boldsymbol{n}} & -T_{\boldsymbol{n}} & & \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} \boldsymbol{u}_S \\ p_S \\ \boldsymbol{u}_D \\ p_D \\ \lambda \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} \boldsymbol{f}_s \\ 0 \\ 0 \\ f_d \\ 0 \end{pmatrix}}_{b}. \tag{6}$$

The block structure of the operator A is mirrored in its implementation in HAZniCS with trace operators implemented using [Kuchta 2021], see Supplementary Material and [Budiša et al. 2022]. The following parameter robust preconditioner for Darcy-Stokes problem (6) is derived in [Holter et al. 2020]:

$$B = \begin{pmatrix} -\mu\nabla\cdot\nabla + DT_{\boldsymbol{\tau}}'T_{\boldsymbol{\tau}} & & & & \\ & \mu^{-1}I & & & \\ & & K^{-1}(I - \nabla\nabla\cdot) & & \\ & & & KI & \\ & & & & \mu^{-1}(-\Delta_\Gamma + I_\Gamma)^{-\frac{1}{2}} + K(-\Delta_\Gamma + I_\Gamma)^{\frac{1}{2}} \end{pmatrix}^{-1}. \tag{7}$$

The non-standard components here are: the first block which involves a lower-order (interfacial) perturbation of the tangential components, and the last block involves a weighted sum of fractional operators of different orders. In Section 3.2 we show how to use HAZniCS methods to implement the preconditioner (7) and, in particular, we discuss the rational approximation method to compute the inverse of the fractional operator. In Section 4.2 we demonstrate its performance.

2.3 Mixed-dimensional modeling of signal propagation in neurons

Interaction of slender bodies with their surrounding is of frequent interest in models of blood flow and oxygen transfer [Berg et al. 2020; Hartung et al. 2021]. It has recently received significant attention as it is a coupling of high dimensional gap (codimension two) which introduces mathematical difficulties [D'Angelo and Quarteroni 2008; Gjerde et al. 2020; Koch et al. 2020; Köppl et al. 2018]. Here, we consider an alternative application in neuroscience. In particular, we apply the coupled $3d$-$1d$ model [Laurino and Zunino 2019] to study electric signaling in neurons and its interaction with the extracellular matrix. The complete model involves a system of PDEs that represents the electrodiffusion and a set of ordinary differential equations (ODE) representing the membrane dynamics. Our focus is on the PDE part that arises from the operator splitting approach to obtain the solution of the full PDE-ODE problem [Jæger et al. 2021].

We use the reduced EMI model [Buccino et al. 2021] that represents the extracellular space as a $3d$ domain and the neuronal body (soma, axons, and dendrites) as one-dimensional curves. The $3d$-$1d$ coupled system states to find extracellular and intracellular potentials (p_3, p_1) that satisfy

$$-\nabla\cdot(\sigma_3\nabla p_3) + \delta_\Gamma \frac{\rho C_m}{\Delta t}(\Pi_\Gamma^\rho p_3 - p_1) = f_3 \qquad \text{in } \Omega, \tag{8a}$$

$$-\frac{\mathrm{d}}{\mathrm{d}s}\cdot(\rho^2\sigma_1\frac{\mathrm{d}}{\mathrm{d}s}p_1) + \frac{\rho C_m}{\Delta t}(p_1 - \Pi_\Gamma^\rho p_3) = f_1 \qquad \text{on } \Gamma. \tag{8b}$$

Here, Ω is a domain in $3d$, while Γ is the $1d$ network of curves representing the neuron by centerlines of soma, axons, and dendrites, see Figure 1. The derivative along Γ is defined in arc length coordinate s. Coupling between

the domains is realized by the averaging operator Π_Γ^ρ which computes the mean of functions in Ω on the idealized cylindrical surface that represents the interface between the dendrites and their surroundings. More precisely, given a point $y \in \Gamma$ and $p : \Omega \to \mathbb{R}$, $\Pi_\Gamma^\rho p : \Gamma \to \mathbb{R}$ is such that $\Pi_\Gamma^\rho p(y) = |C_\rho(y)|^{-1} \int_{C_\rho(y)} u \, dl$ where $C_\rho(y)$ is a circle centered at y with radius ρ in the plane whose normal ν is given by the tangent to Γ at y, cf. Figure 1. That is, ρ represents the radius of a neuron segment and, as such, typically varies in space. However, for simplicity of the presentation, we assume ρ to be constant. Moreover, by δ_Γ we denote the Dirac measure of Γ. The term $\frac{\rho C_m}{\Delta t}(p_1 - \Pi_\Gamma^\rho p_3)$ represents the electric current flow exchange between the domains across dimensions due to the potential differences, with Δt being the time step size. The parameters σ_3, σ_1 and C_m represent the extracellular and intracellular conductivity and the membrane capacitance, respectively. Assuming $\partial\Omega_D \cup \partial\Omega_N = \partial\Omega \backslash \Gamma$ and $\partial\Omega_D \cap \partial\Omega_N = \emptyset$, we also impose the boundary conditions

$$p_3 = g_3 \qquad \text{on } \partial\Omega_D, \tag{9a}$$

$$-\sigma_3 \nabla p_3 \cdot \boldsymbol{n} = 0 \qquad \text{on } \partial\Omega_N, \tag{9b}$$

$$-\rho^2 \sigma_1 \nabla p_1 \cdot \boldsymbol{n} = 0 \qquad \text{on } \partial\Gamma. \tag{9c}$$

As previously, we relate a linear system of equations to (8). Let $Q_3 \subset H^1(\Omega)$ and $Q_1 \subset H^1(\Gamma)$ be conforming finite element spaces (e.g. $\mathbb{P}_1$) on the shape-regular triangulation of Ω and Γ, respectively. The discrete linear system corresponding to (8) states to find $(p_3, p_1) \in Q_3 \times Q_1$ such that

$$\Bigg[\underbrace{\begin{pmatrix} -\sigma_3 \Delta_\Omega & \\ & -\rho^2 \sigma_1 \Delta_\Gamma \end{pmatrix}}_{A_D} + \tilde{\rho}_t \underbrace{\begin{pmatrix} \Pi_\Gamma^{\rho\prime} \\ -I \end{pmatrix} \begin{pmatrix} \Pi_\Gamma^\rho & -I \end{pmatrix}}_{M} \Bigg] \underbrace{\begin{pmatrix} p_3 \\ p_1 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} f_3 \\ f_1 \end{pmatrix}}_{b}, \tag{10}$$

with $\tilde{\rho}_t = \frac{\rho C_m}{\Delta t}$ and the system operator $A = A_D + \tilde{\rho}_t M$. We call linear systems such as (10) *metric-perturbed* problems, and their implementation in HAZniCS is available in [Budiša et al. 2022] and Supplementary Material.

The operator A is symmetric positive definite and we can use the preconditioned conjugate gradient (CG) method to solve the system (10). However, we observe that the bilinear form represented by M is degenerate. That is, for very large values of the coupling parameter $\tilde{\rho}_t$, the semi-definite coupling part M dominates and the system becomes nearly singular. We identify that the operator M induces an L^2-based *metric space* $\mathcal{M}(\Gamma) = \{(q_3, q_1) \in Q_3 \times Q_1 : \int_\Gamma (\Pi_\Gamma q_3 - q_1)^2 < \infty\}$ and the singular part is related to the kernel of the coupling operator, that is $\ker(M) = \{(q_3, q_1) \in Q_3 \times Q_1 : \Pi_\Gamma q_3 - q_1 = 0\}$ which can be a large subspace of the solution space. Consequently, the condition number of the system grows rapidly with increasing $\tilde{\rho}_t$, which results in slow convergence of the CG solver, even when using the standard AMG method as the preconditioner as in Section 2.1. We remark that [Cerroni et al. 2019] demonstrate that (standard, smoothed aggregation) AMG leads to robust solvers when the coupling is weak ($\tilde{\rho}_t \ll 1$). In Section 3.3 we demonstrate how to implement and solve the system (10) with HAZniCS methods based on the AMG with specialized block Schwarz smoothers. In Section 4.3 we showcase some key performance points of the solver.

3 IMPLEMENTATION

The software module HAZniCS combines several libraries, each providing a key functionality for multiphysics simulations. The main components include:

(i) HAZmath [Adler et al. 2009] - a finite element, graph, and solver library built in C;
(ii) FEniCS [Logg et al. 2012] - a computing platform in Python for solving PDEs;
(iii) cbc.block [Mardal and Haga 2012] - an extension to FEniCS that enables assembling and solving block-partitioned problems;

(iv) FEniCS$_{ii}$ [Kuchta 2021] - an extension to FEniCS for assembling systems of equations posed on domains of different dimensionality.

In HAZniCS, each preconditioning method mentioned in the Section 2 is implemented in HAZmath as a C function with the same signature - it takes in a vector (an array of `double` values), applies a set of operations and returns a solution vector. To be able to use it in Python, the HAZniCS Python library is generated using SWIG [Beazley 1996] and, in turn, can be imported simply as `import haznics`. In the following code snippets, we demonstrate how this interface is built.

First, we note that while Python and FEniCS use memory management systems, HAZmath does not and users are required to keep track of the memory themselves. As such, any object transferred between the two systems is copied to avoid that the memory systems of Python or FEniCS delete data that is in use in HAZmath. We also remark that, although SWIG contains tools for making C code appear Pythonic (i.e. distinguishing between function input and output) we have opted for making the Python interface a direct translation of the underlying C code.

For each preconditioner, HAZniCS stores two functions - a setup and an application function. The setup functions take in different variables depending on the type of the preconditioner but always return a pointer to HAZmath data type `precond` of a general preconditioner. This data type has two components: preconditioner data structure `data` and matrix-vector operation function `fct()`, see Listing 1, where type `REAL` is a macro of the standard C type `double`.

```
typedef struct {
  void *data;
  void (*fct)(REAL *, REAL *, void *);
} precond;
```

Listing 1. HAZmath structure for type precond.

During setup, HAZmath saves all matrices, vectors and parameters necessary for applying the preconditioner to a data structure, and points to the function that executes the matrix-vector application algorithm. All preconditioner functions have the same signature - they take in two arrays of `REAL` values (input and output vectors) and data related to the matrix-vector operation as `void*`. Note that different preconditioner algorithms require different information to be saved, thus the different preconditioner data structure types are passed as void pointers and then type cast back to original data types in the corresponding (preconditioner-specific) application function, cf. for example Listing 6.

Using the generated HAZniCS Python library, we wrap the HAZmath preconditioner functions as class methods in cbc.block. In this way, efficient HAZmath preconditioners (in C) can be used with FEniCS (or PETSc) operators and cbc.block iterative methods (in Python) in a code that is easily readable and simple to utilize. We remind that, in HAZmath, all preconditioner functions have the same signature. On the other hand, in any cbc.block iterative method, all preconditioners are applied through a matrix-vector product method `matvec()`. Hence, we define a base class `Precond` equipped with a `matvec()` method designed specifically for call of HAZmath preconditioners, see Listing 2. The class is derived from the cbc.block data type `block_base`, making the HAZmath preconditioners fully integrated with other classes and methods of the cbc.block library.

```
class Precond(block_base):
  #[...]
  def matvec(self, b):
    #[...]
    # create solution vector
    x = self.A.create_vec(dim=1)
    x = df.Vector(df.MPI.comm_self, x.size())
```

```
    #[...]
    # convert rhs and lhs to numpy arrays
    b_np = b[:]
    x_np = x[:]
    # apply the preconditioner (solution saved in x_np)
    haznics.apply_precond(b_np, x_np, self.precond)
    # convert x_np to GenericVector
    x.set_local(x_np)
    return x
```

Listing 2. Baseclass Precond with `matvec()` implemented in HAZmath backend of cbc.block.

The preconditioner's matrix-vector operation is then applied through a simple HAZmath function `apply_precond()`, see Listing 3. It uses the data and a preconditioner-specific matrix-vector function that are passed in the input variable `precond *pc`, cf. Listing 1. Note that the input vectors (`numpy` arrays) need to be cast to C arrays (`REAL *`) to perform the matrix-vector operation, which is done using typemaps in SWIG. A snippet of the typemap is available in Supplementary Material.

```
void apply_precond(REAL *r, REAL *z, precond *pc) {
  pc->fct(r, z, pc->data);
}
```

Listing 3. HAZmath function `apply_precond()`.

In the following, we detail the implementation of the preconditioners used in Section 2. In Section 3.1 and Section 3.2 we show the AMG and the rational approximation preconditioners employed through the cbc.block extension. Alternatively, we can directly call HAZmath functions within FEniCS without relying on cbc.block, since we already have a compiled Python library HAZniCS. This use case is shown in Section 3.3 where we describe solvers for metric-perturbed problems.

3.1 Algebraic multigrid method

As the main preconditioning routine, we use the Algebraic MultiGrid Method (AMG) [Brandt et al. 1982], which constructs a multilevel hierarchy of vector spaces, each of which is responsible for correcting different components of the error (see also [Xu and Zikatanov 2017] and the references therein). Specifically, our approach is based on the *Unsmoothed Aggregation* (UA-AMG) [Blaheta 1986; Marek 1991; Vakhutinsky et al. 1979] and the *Smoothed Aggregation* (SA-AMG) method [Hu et al. 2016; Míka and Vaněk 1992; Míka and Vaněk 1992; Vaněk et al. 1996, 1998]. The UA-AMG algorithms in HAZmath are based on graph matching (or pairwise aggregation) and can be found in [Hu et al. 2019; Kim et al. 2003].

```
from haznics import AMG
# AMG setup parameters
params = {
    "AMG_type": haznics.UA_AMG,
    "cycle_type": haznics.NL_AMLI_CYCLE,
    "smoother": haznics.SMOOTHER_GS,
    "coarse_solver": haznics.DIRECT,
    "aggregation_type": haznics.VMB,
    "max_aggregation": 100,
}
# Solver setup
B = AMG(A, params)
Ainv = ConjGrad(A, precond=B, tolerance=1e-10)
```

```
# Solve
x = Ainv * b
```

Listing 4. Call of the AMG preconditioner for the linear elliptic problem (1). Complete code can be found in script `HAZniCS-examples/demo_elliptic_test.py`.

In Listing 4 we showcase how to use the AMG from HAZmath as the preconditioner in FEniCS-related examples. We import the preconditioner class `AMG` from cbc.block, implementation of which is shown in Listing 5. It takes the coefficient matrix A and an optional dictionary of setup parameters. Such parameters are set through HAZmath macros and are integrated within the HAZniCS Python library through a dictionary. For example, we can specify the type of the AMG method we will apply using the keyword `"AMG_type"` and the value `haznics.UA_AMG`. Other listed keywords determine `"cycle_type"` (cycling algorithm), `"smoother"` (type of smoother), `"coarse_solver"` (coarse grid solver), `"max_aggregation"` (maximum number of vertices in an aggregate), and `"aggregation_type"` (type of aggregation). Full list of parameters and values of macros can be found in HAZmath files `include/params.h` and `include/macro.h`, respectively.

```
class AMG(Precond):
    def __init__(self, A, parameters=None):
        # change data type for the matrix (to dCSRmat pointer)
        A_ptr = PETSc_to_dCSRmat(A)
        # initialize amg parameters (AMG_param pointer)
        amgparam = haznics.AMG_param()
        # set extra amg parameters
        if parameters:
            haznics.param_amg_set_dict(parameters, amgparam)
        # set AMG preconditioner
        precond = haznics.create_precond_amg(A_ptr, amgparam)
        #[...]
        Precond.__init__(self, A, "AMG", parameters, amgparam, precond)
```

Listing 5. Preconditioner class `AMG` implemented in HAZmath backend of cbc.block.

We show an example of the implementation of the AMG preconditioner class in Listing 5. Before calling the preconditioner setup function, some input FEniCS data types need to be converted to HAZmath data types. For example, an auxiliary function `PETSC_to_dCSRmat()` converts FEniCS or PETSc matrices to HAZmath matrices. This conversion is simple, as FEniCS/PETSc and HAZmath utilize compressed sparse row (CSR) format.

Coming back to Listing 4, the AMG preconditioner is passed to the CG iterative solver `ConjGrad` from cbc.block to act on the residual in each iteration. As shown in the previous section, the application of the preconditioner is made as a matrix-vector operation, which in the case of the AMG method corresponds to the function `precond_amg()` stated in Listing 6. It reads the AMG setup data through the variable `pcdata->mgl_data`, sets up the right-hand side vector (the residual `r` of the outer iterative method) and initializes the solution vector, applies the AMG algorithm through the function `mgcycle()` and returns the computed solution via the increment variable `z`.

```
void precond_amg(REAL *r, REAL *z, void *data) {
  precond_data *pcdata=(precond_data *)data; // data for the preconditioner
  const INT m = pcdata->mgl_data[0].A.row; // general size of the system
  const INT maxit = pcdata->maxit; // how many times to apply AMG
  INT i;

  AMG_param amgparam; param_amg_init(&amgparam);
  param_prec_to_amg(&amgparam, pcdata); // set up AMG parameters
```

```
    AMG_data *mgl = pcdata->mgl_data; // data for the AMG
    mgl->b.row = m; array_cp(m, r, mgl->b.val); // residual is the rhs
    mgl->x.row = m; dvec_set(m, &mgl->x, 0.0);

    for (i = 0; i < maxit; ++i) mgcycle(mgl, &amgparam); // apply AMG

    array_cp(m, mgl->x.val, z); // copy the result to z
}
```

Listing 6. Implementation of the AMG preconditioner in HAZmath.

Using HAZmath's implementation of the AMG method in function `mgcycle()` gives the flexibility to apply and modify the algorithm to other relevant applications. The next two sections present how we use it in methods that approximate inverses of fractional and metric-perturbed operators.

3.2 Solvers for indefinite problems with fractional operators

Implementation of the Darcy-Stokes preconditioner (7) within HAZniCS is given in Listing 7. First, we recognize that the preconditioner extracts the relevant block from the operator A to construct the Stokes velocity preconditioner, while the remaining inner product operators are assembled (as they are not part of A, cf. system (6)). The option to extract or assemble the (auxiliary) operators to define a preconditioner is a powerful feature of HAZniCS/cbc.block. Hence, in Listing 7, we use scalable algorithms from HAZniCS and PETSc [Balay et al. 2022] to approximate the inverses of each block separately. AMG schemes are used for the inverses of operators on $\boldsymbol{V}_S$ and $\boldsymbol{V}_D$ where, particularly in the latter, the HAZniCS preconditioner class `HXDiv` implements the auxiliary space method [Hiptmair and Xu 2007] for $H(\mathrm{div})$ problems [Kolev and Vassilevski 2012]. Inverses of operators representing L^2 inner products on the pressure spaces are realized via simple iterative schemes such as the symmetric successive over-relaxation (SSOR). Finally, the preconditioner for the Lagrange multiplier, which involves the inverse of a sum of fractional operators, is computed with the rational approximation method that employs AMG (`mgcycle()`) internally.

We discuss in the following how to use and implement rational approximation [Hofreither 2020] that acts as an application of the inverse of a fractional operator.

3.2.1 Rational approximation. Let $s, t \in [-1, 1]$ and $\alpha, \beta \geq 0$. The basic idea is to find a rational function approximating $f(x) = (\alpha x^s + \beta x^t)^{-1}$, $x > 0$, that is,

$$(\alpha x^s + \beta x^t)^{-1} \approx R(x) = \frac{P_{k'}(x)}{Q_k(x)}, \tag{11}$$

where $P_{k'}$ and Q_k are polynomials of degree k' and k, respectively. Assuming $k' \leq k$, the rational function can be given in partial fraction form $R(x) = c_0 + \sum_{i=1}^{n_p} \frac{c_i}{x - p_i}$ for $c_0 \in \mathbb{R}$, $c_i, p_i \in \mathbb{C}$, $i = 1, 2, \ldots, n_p$. With a slight abuse of notation, denote with A a symmetric positive definite operator. Then, the rational function $R(\cdot)$ can be used to approximate $f(A)$ as follows,

$$z = f(A)r \approx c_0 r + \sum_{i=1}^{n_p} c_i \left(A - p_i I\right)^{-1} r. \tag{12}$$

The overall algorithm is shown in Algorithm 1. In our case, the operator A is a discretization of the Laplacian operator $-\Delta$, and I is the discrete operator of the L^2 inner product. Therefore, the equations in Step 1 of Algorithm 1 can be viewed as discretizations of the shifted Laplacian problems $-\Delta\, w_i - p_i\, w_i = r$. For real non-positive poles, the problem is SPD, so we may define fractions or functions of the operator $-\Delta - p_i I$.

```
from block.algebraic.hazmath import RA, AMG, HXDiv
from block.algebraic.petsc import SOR
# Composite function space
VS, QS, VD, QD, Q = W
# Define SPD operators defining inner products on the spaces
# Stokes velocity block is taken from the system matrix
B0 = A[0][0]
# L^2 inner product on QS
B1 = assemble((1/mu)*inner(TrialFunction(QS), TestFunction(QS))*dx)
# H(div) inner product on VD
uD, vD = TrialFunction(VD), TestFunction(VD)
B2 = assemble((1 / K) * (inner(u2, v2) * dx + inner(div(u2), div(v2)) * dx))
# L^2 inner product on QD
B3 = assemble(K*inner(TrialFunction(QD), TestFunction(QD))*dx)
# Lagrange Multiplier requires -\Delta + I and I on Q
p, q = TrialFunction(Q), TestFunction(Q)
h = CellDiameter(bmesh)
Deltag = assemble(avg(h)**(-1) * dot(jump(p), jump(q)) * dS + inner(p, q) * dx)
Ig = assemble(inner(p, q) * dx)  # in Delta_g we use DG discretization
# For inversion we require parameters for rational approximation (RA)
params = {'coefs': [1. / mu(0), K(0)], 'pwrs': [-0.5, 0.5], '#[...]'}
B4 = RA(Deltag, Ig, parameters=params)

# Define the approximate preconditioner
B = block_diag_mat([AMG(B0), SOR(B1), HXDiv(B2), SOR(B3), B4])
```

Listing 7. Scalable implementation of Darcy-Stokes preconditioner (7). Complete code can be found in scripts `HAZniCS-examples/demo_darcy_stokes*.py`.

Algorithm 1 Compute $z = f(A)r$ using rational approximation.

1: Solve for w_i: $(A - p_i I)\, w_i = r, \quad i = 1, 2, \ldots, n_p$.

2: Compute: $z = c_0 r + \sum_{i=1}^{n_p} c_i w_i$

As remarked earlier, the finite element matrices are always associated with mappings between nodal and dual representations. In the following, we describe such representations for the preconditioners of interest. Let A be the stiffness matrix corresponding to a discretization of $(-\Delta)$ and M be the corresponding mass matrix. Consider the following generalized eigenvalue problem

$$\mathsf{AU} = \mathsf{MU}\Lambda, \quad \mathsf{U}^T\mathsf{MU} = \mathsf{I} \quad \Longrightarrow \quad \mathsf{U}^T\mathsf{AU} = \Lambda. \tag{13}$$

For any continuous function $F(x)$, $x \in [0, \rho]$ we define

$$F(\mathsf{A}) \coloneqq \mathsf{MU}f(\Lambda)\mathsf{U}^T\mathsf{M}, \tag{14}$$

where $\rho \coloneqq \rho\left(\mathsf{M}^{-1}\mathsf{A}\right)$ is the spectral radius of the matrix $\mathsf{M}^{-1}\mathsf{A}$. We would like to approximate $f(\mathsf{A}) = (F(\mathsf{A}))^{-1}$ using the rational approximation $R(x)$ of $f(x) = \frac{1}{F(x)}$. If we have a function $g(t) = f(\rho t)$ defined on the interval $[0, 1]$ and $r(t)$ is the best rational approximation to $g(t)$, then

$$f(x) \approx R(x) = r\left(\frac{x}{\rho}\right) \approx g\left(\frac{x}{\rho}\right), \quad r\left(\frac{x}{\rho}\right) = c_0 + \sum_{i=1}^{n_p} \frac{c_i}{\frac{x}{\rho} - p_i}. \tag{15}$$

Therefore, if we know c_i and p_i for $g(t)$ on the interval $[0, 1]$, the residues and poles for $f(x) = f(\rho t)$ are ρc_i and ρp_i, respectively.

Using that, (13) and (14), the rational approximation of $f(\mathsf{A})$ is a *dual* to *nodal* mapping as follows

$$f(\mathsf{A}) \approx c_0 \mathsf{M}^{-1} + \sum_{i=1}^{n_p} \rho c_i \left(\mathsf{A} - \rho p_i \mathsf{M}\right)^{-1}. \tag{16}$$

In summary, to apply the rational approximation, we need to find solvers to apply M^{-1} and each $(\mathsf{A} - \rho p_i \mathsf{M})^{-1}$. If $p_i \in \mathbb{R}$, $p_i \leq 0$, we end up solving a series of elliptic problems where multigrid methods are very efficient. As mentioned in Section 3.1, the HAZmath library contains several fast-performing implementations of the AMG method, such as SA-AMG and UA-AMG methods.

Furthermore, many methods compute the coefficients c_i and p_i, cf. an overview in [Hofreither 2020]. In HAZmath, we have implemented the Adaptive Antoulas-Anderson (AAA) algorithm proposed in [Nakatsukasa et al. 2018]. The AAA method is based on a representation of the rational approximation in barycentric form and greedy selection of interpolation points. In most cases, this approach leads to $p_i \leq 0$, so we can use the AMG method to solve each problem in Step 1 of Algorithm 1.

```
class RA(Precond):
  def __init__(self, A, M, parameters=None):
    # change data type for the matrices (to dCSRmat pointer)
    A_ptr, M_ptr = map(PETSc_to_dCSRmat, (A, M))
    # initialize amg parameters (AMG_param pointer)
    amgparam = haznics.AMG_param()
    #[...]
    haznics.param_amg_set_dict(parameters, amgparam)
    # get scalings
    scaling_a = 1. / A.norm("linf")
    scaling_m = 1. / df.as_backend_type(M).mat().getDiagonal().min()[1]
    # get coefs and powers
    alpha, beta = parameters['coefs']
    s_power, t_power = parameters['pwrs']
    # set RA preconditioner #
    precond = haznics.create_precond_ra(A_ptr, M_ptr, s_power, t_power,
                                        alpha, beta, scaling_a, scaling_m,
                                        amgparam)
    # [...]
    Precond.__init__(self, A, "RA", parameters, precond)
```

Listing 8. Class RA implemented in HAZmath backend of cbc.block.

In the demo examples `HAZniCS-examples/demo_darcy_stokes*.py`, see also Listing 7 and Supplementary Material, we use the rational approximation method from HAZmath for the fractional operator block in (7). First, we import the preconditioner class RA representing the rational approximation method from the cbc.block backend designated for HAZmath methods. It takes in two matrices, A and M, that are the discretizations of H^1 and L^2 inner products on the solution domain. It also takes in an optional dictionary of parameters that, among others, specify weights α, β and fractional powers s, t. The constructor of the class RA sets up the preconditioner data, see Listing 8.

That is, it computes:

- the coefficients c_i, p_i with the AAA algorithm based on matrices A and M and parameters α, β in keyword `'coefs'` and s, t in keyword `'pwrs'`;
- AMG levels for each $\mathsf{A} - p_i\mathsf{M}$ based on optional additional parameters in the `parameters` dictionary.

These two steps are performed in the function `create_precond_ra()` in HAZmath.

Additionally, we compute the upper bound on $\rho\left(M^{-1}A\right)$. For example, in case of $\mathbb{P}_1$ finite elements, the bound depends on the matrix norms $\|M^{-1}\|_\infty$ and $\|A\|_\infty$ and the topological dimension of the problem, see [Budiša et al. 2023b]. Hence, the function `create_precond_ra()` takes scaling parameters to approximate the spectral radius. In practice, M^{-1} scales as (at least) h^{-1}, so the topological dimension is not an important factor in the scalings.

The rational approximation preconditioner is then applied in each iteration through a matrix-vector function, as explained at the beginning of Section 3. In this case, the matrix-vector function is the HAZmath function `precond_ra_fenics()` that applies the two steps from Algorithm 1. In Listing 9 we show the implementation snippet of the key parts of the preconditioner algorithm.

```
void precond_ra_fenics(REAL *r, REAL *z, void *data) {
  //[...]
  // z = z + residues[0] * M^{-1} r
  if(fabs(residues->val[0]) > 0.) {
    status = dcsr_pcg(scaled_M, &r_vec, &z_vec, &pc_scaled_M, 1e-6, 100, 1, 0);
  }
  array_ax(n, residues->val[0], z_vec.val);
  // [...]
  for(i = 0; i < npoles; ++i) {
    //[...]
    dvec_set(update.row, &update, 0.0);
    // solve (A - poles[i] * M) update = r
    status = dcsr_pcg(&(mgl[i][0].A), &r_vec, &update, &pc_frac_A,1e-6,100,1,0);
    //[...]
    // z = z + residues[i+1]*update
    array_axpy(n, residues->val[i+1], update.val, z_vec.val);
  }
}
```

Listing 9. HAZmath function `precond_ra_fenics()`.

3.3 Solvers for interface metric-perturbed problems

We continue with presenting the implementation of the solver for the $3d$-$1d$ coupled problem in Section 2.3. Previously, we bridged the HAZmath and FEniCS libraries via a class of preconditioners implemented in cbc.block, while here we introduce an alternative way to use HAZmath solvers. In the file `demo_3d1d.py` we have specified the block problem (10) using FEniCS extensions FEniCS$_{ii}$ and cbc.block, see Supplementary Material and [Budiša et al. 2022]. Next, we display in Listing 10 the steps necessary to use HAZmath solver for this block problem directly through the library `haznics`. The listing consists of two parts: data conversion and calling the solver wrapper function.

First, after assembly, the system matrix A and the right hand side b are of type `block_mat` and `block_vec`, respectively. We convert them to HAZmath data types `dvector` (vector of `double` values) and `block_dCSRmat` (block CSR matrix) so we can use them in the solver called through the function `fenics_metric_amg_solver()`. This auxiliary HAZniCS function acts as an intermediary to set solver data and parameters and to run the solver. It can be found in file `src/haznics/helper.c`.

Furthermore, unless we want to use default values, we can set relevant parameters for the HAZmath solver, such as the tolerance of the iterative method or the type of the preconditioner. This can be done by creating an input file that passes the specific parameters to HAZmath to an `input_param` type variable. The file is then read within the solver wrapper function. A snippet of the input file `input_metric.dat` for the problem (10) can be found in Supplementary Material.

```
# convert vectors
bb = ii_convert(b)
b_np = bb[:]
bhaz = haznics.create_dvector(b_np)
xhaz = haznics.dvec_create_p(n)
# convert matrices; A = AD + rho * M
Ahaz = block_to_haz(A)
Mhaz = block_to_haz(M)
ADhaz = block_to_haz(AD)

# call solver
niters = haznics.fenics_metric_amg_solver(Ahaz, bhaz, xhaz, ADhaz, Mhaz)
```

Listing 10. Call of HAZniCS solver for the $3d$-$1d$ coupled system (10). Complete code can be found in script `HAZniCS-examples/demo_3d1d.py`.

This setup allows to apply the metric-perturbed AMG solver. We recall that we have chosen to solve the $3d$-$1d$ problem (10) by the CG method preconditioned with AMG that uses block Schwarz smoothers to obtain robustness in the coupling parameter $\tilde{\rho}_t \gg 1$. The HAZmath implementation of that solver has a slight modification that uses a combination of the block Schwarz and Gauss-Seidel smoothers. We give a few details on the algorithm and its implementation in the following section, while the full code is available in file `src/solver/itsolver.c`.

3.3.1 Metric-perturbed algebraic multigrid method. Let us go back to the system (10) and set $V = Q_3 \times Q_1$. The general subspace correction method looks for a stable space decomposition

$$V = V_0 + V_1 + \ldots + V_J \tag{17}$$

to divide solving the system on the whole space V to solving smaller problems on each subspace and summing up the contributions in additive or multiplicative fashion [Xu 1992; Xu and Zikatanov 2002]. Furthermore, the following condition from [Lee et al. 2007] is sufficient to obtain a robust subspace correction method to solve nearly singular system such as (10):

$$\mathrm{Ker}(M) \cap V = (\mathrm{Ker}(M) \cap V_0) + (\mathrm{Ker}(M) \cap V_1) + \ldots + (\mathrm{Ker}(M) \cap V_J). \tag{18}$$

We employ this space decomposition to create a robust AMG method where V_0 represents the coarse space and $V_i,\ i = 1, \ldots, J$, define a Schwarz-type smoother. By robustness, we imply that the convergence of the method is independent of the values of the coupling parameter $\tilde{\rho}_t$ and mesh parameter h. To construct subspace splitting satisfying (18), it is necessary to choose the subspaces so that the following holds: for each element of a frame spanning the kernel of M, there exists a subspace V_i containing this frame element. Notice that this requirement does not assume that the frame element is known, but rather that the subspace where this element is contained is known.

Algorithm 2 Compute $z = Br$ using metric-perturbed AMG

Require: Given r and $z \leftarrow 0$

1: Solve on the interface using forward Schwarz smoother: $z \leftarrow z + \Pi_\Gamma^\rho B_{\mathrm{Schwarz}} \Pi_\Gamma' r$
2: Solve on the whole space using AMG method: $z \leftarrow z + B_{\mathrm{AMG}}(r - Az)$
3: Solve on the interface using backward Schwarz smoother: $z \leftarrow z + \Pi_\Gamma^\rho B'_{\mathrm{Schwarz}} \Pi_\Gamma' (r - Az)$

The metric-perturbed AMG method is given in the Algorithm 2. We can see that B is defined as

$$I - BA := (I - \Pi_\Gamma^\rho B'_{\mathrm{Schwarz}} \Pi_\Gamma' A)(I - B_{\mathrm{AMG}} A)(I - \Pi_\Gamma^\rho B_{\mathrm{Schwarz}} \Pi_\Gamma' A).$$

It is easy to see that B is symmetric and, following the theory developed in [Hu et al. 2013], B is also positive definite if B_{AMG} is symmetric positive definite and B_{Schwarz} is nonexpansive. Therefore, it can be used as a preconditioner for the CG method. This preconditioner is implemented in HAZmath and its excerpt from the function is given in Listing 11.

```
void precond_bdcsr_metric_amg_symmetric(REAL *r, REAL *z, void *data)  {
  //[...]
  // Schwarz method on the interface part
  Schwarz_param *schwarz_param = predata->schwarz_param;
  Schwarz_data *schwarz_data = predata->schwarz_data;
  smoother_dcsr_Schwarz_forward(schwarz_data, schwarz_param, &zz, &rr);
  //[...]
  // AMG solve on the whole matrix
  AMG_data_bdcsr *mgl = predata->mgl_data;
  mgl->b.row  total_row; array_cp(total_row, r, mgl->b.val);
  mgl->x.row = total_col; array_cp(total_row, z, mgl->x.val);
  for ( i=maxit; i--; ) mgcycle_bdcsr(mgl,&amgparam);
  //[...]
  // Schwarz method on the interface part
  smoother_dcsr_Schwarz_backward(schwarz_data, schwarz_param, &zz, &rr);
  //[...]
}
```

Listing 11. Implementation of the metric AMG preconditioner in Algorithm 2.

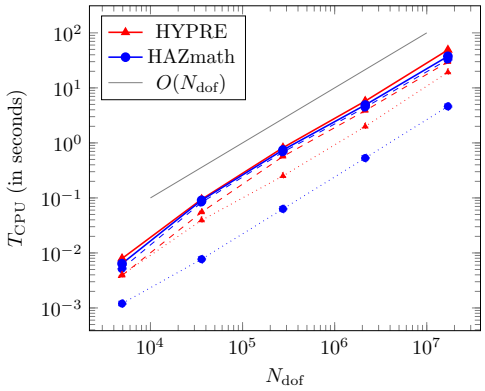
4 RESULTS

In this section, we show the performance of the solvers and preconditioners developed for the examples in Section 2. We recall that their complete code can be found in [Budiša et al. 2022]. Due to limited data availability, we use meshes that have matching degrees of freedom and conform to the interface, although we note that that all examples can work with unfitted meshes. What is needed is the interface operator which relates the degrees of freedom from one mesh to the other. Such an operator (averaging, interpolation, etc.) is always represented by a sparse matrix, and it enables us to recover and eliminate the algebraic low frequencies efficiently without additional information about the geometry of the mesh.

4.1 Linear elliptic problem

We use the 3d elliptic problem (1) to compare the HAZniCS solvers to already established solver libraries. This way, we demonstrate that HAZniCS, specifically the AMG solver within, shows a fast and reliable performance when solving common PDE problems. For the comparison, we use the AMG method BoomerAMG from the HYPRE library [Falgout and Yang 2002] of scalable linear solvers and multigrid methods that are already integrated within FEniCS software through PETSc. We note that all the computations are performed in serial on a workstation with an Intel® Core™ i7-1165G7 @ 2.80GHz (8 cores) CPU and 40GB of RAM.

The results are given in Figure 2. It is clear that the AMG methods of HYPRE and HAZmath show similar performance. While the HYPRE BoomerAMG method gives fewer total CG iterations and consequently less solving time, the setup of the HAZmath UA-AMG method is multiple times faster while still taking comparable solving time. Therefore, we are confident about using the methods from HAZniCS in our multiphysics solvers, namely the HAZmath's AMG method as a component of the rational approximation and metric-perturbed preconditioners from Sections 3.2 and 3.3.



N_{dof}	N_{iter} HAZmath	HYPRE
729	9	7
4913	10	8
35937	11	8
274625	11	9
2146689	12	9
16974593	12	9

Fig. 2 & Table 1. Performance of the CG method preconditioned with either HAZmath AMG or HYPRE AMG, with regards to the number of degrees of freedom N_{dof} and up to relative residual tolerance 10^{-6}. Results are obtained by running `HAZniCS-examples/demo_elliptic_test.py`. (Left) The setup (dotted line), solve (dashed line) and total (full line) CPU time T_{CPU} required to solve (1). The blue data points represent HAZmath AMG preconditioner, while the red data points represent HYPRE AMG preconditioner. (Right) Number of iterations (N_{iter}) of the CG method preconditioned with either HAZmath AMG or HYPRE AMG.

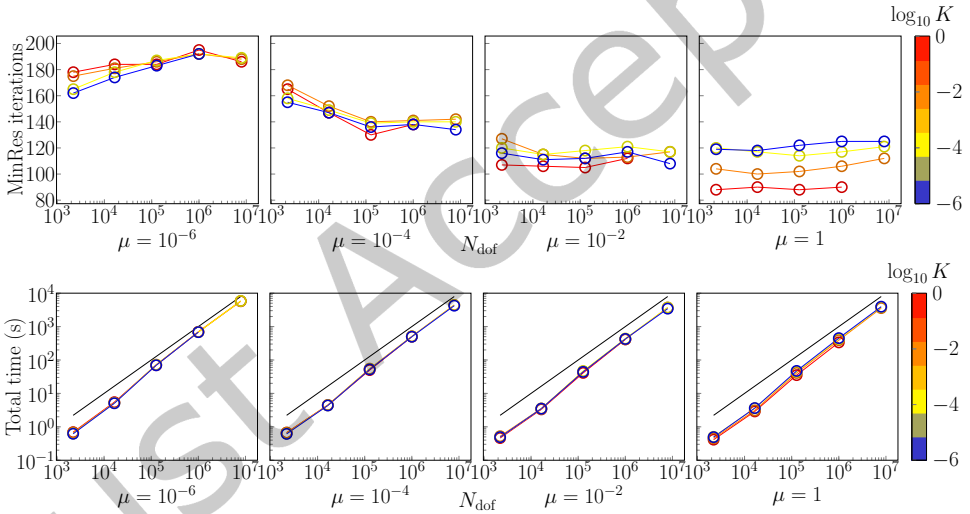


Fig. 3. Performance of Darcy-Stokes preconditioner (7) implemented in Listing 7. Discretization by $(\mathbb{CR}_1-\mathbb{P}_0)-(\mathbb{RT}_0-\mathbb{P}_0)-\mathbb{P}_0$ elements with $D = 0.1$. (Top) Number of MinRes iterations until convergence for different values of μ, K and mesh sizes. (Bottom) Total solution time for solving (6) including the setup time of the preconditioner. Black line indicates linear scaling. Results are obtained by running `HAZniCS-examples/demo_darcy_stokes_3d_flat.py`.

4.2 Darcy-Stokes problem

To demonstrate the performance of the rational approximation algorithms of HAZniCS, we next focus on the Darcy-Stokes system (6) and its preconditioner (7). Implementation of the preconditioner in HAZniCS can be found in Listing 7, and we recall that we utilize multilevel methods for the Stokes velocity and Darcy flux blocks while the multiplier block uses rational approximation detailed in Section 3.2.

Let us first showcase the robustness and scalability of implementing the Darcy-Stokes preconditioner. Here we focus on the (more challenging) case $\Omega_S, \Omega_D \subset \mathbb{R}^3$ while results for a similar study in two dimensions are given

in Supplementary Material. Let now $\Omega_S = \left[0, \frac{1}{2}\right] \times [0,1]^2$ and $\Omega_D = \left[\frac{1}{2}, 1\right] \times [0,1]^2$. We consider discretization of (6) by (stabilized, cf. Supplementary Material), $\mathbb{CR}_1$-$\mathbb{P}_0$ elements in the Stokes domain, $\mathbb{RT}_0$-$\mathbb{P}_0$ elements in the Darcy domain and $\mathbb{P}_0$ elements on the interface. Using gradually refined meshes of $\Omega_S \cup \Omega_D$, which match on the interface Γ, the choice of elements leads to linear systems with $2 \cdot 10^3 < N_{\text{dof}} < 11 \cdot 10^6$. Furthermore, we shall vary the model parameters such that $10^{-6} \leq \mu, K \leq 1$ while $D = 0.1$ is fixed.

The performance of the preconditioner is summarized in Figure 3, where we list the dependence of the solution time and the number of MinRes iterations on mesh size and model parameters. Here the convergence criterion is the reduction of the preconditioned residual norm by 10^{12}. Moreover, the tolerance in the rational approximation is set to 10^{-12} yielding roughly $n_p \approx 20$ poles in (12). However, numerical experiments [Budiša et al. 2023b] suggest that a less accurate approximation, leading to as little as 6 poles, could be sufficient. In Figure 3, it can be seen that iteration counts are bounded in the parameters, that is, (7) defines a parameter robust Darcy-Stokes preconditioner. Moreover, the implementation in Listing 7 leads to optimal, $O(N_{\text{dof}})$, scaling.

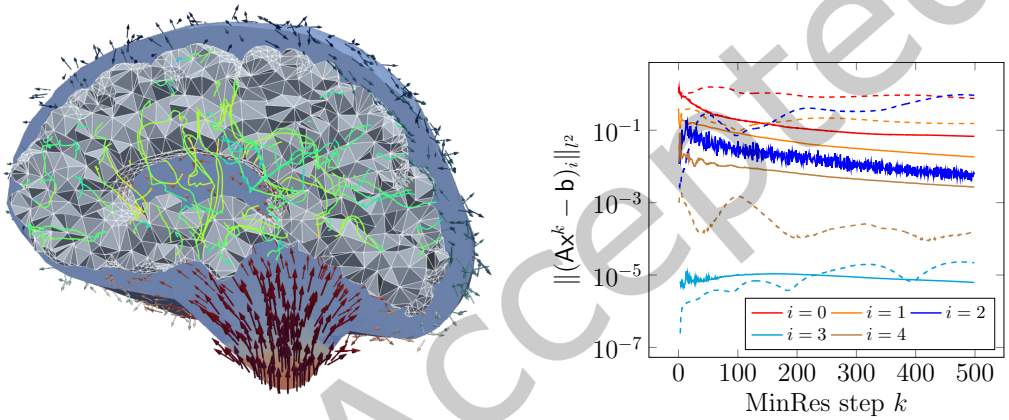


Fig. 4. Darcy-Stokes model on realistic brain geometry. (Left) Clip of the solution field when no-slip boundary conditions are considered everywhere except on the small region on the base (cf. larger pressure in red). Here, the traction locally increases pressure and induces flow in the Stokes domain. Pressure in the Darcy domain is rather uniform. Flow in the Darcy domain is visualized by green streamlines. (Right) Convergence of the solution components (denoted by $0 \leq i \leq 4$ for subspaces $\boldsymbol{V}_S$, Q_S, $\boldsymbol{V}_D$, Q_D and Λ) of (6) when using preconditioner (7) (solid lines). Dashed lines show (diverging) behavior when using a simpler preconditioner which utilized $(-\Delta + I)^{1/2}$ on the interface. Results are obtained by running `HAZniCS-examples/demo_darcy_stokes_brain.py`.

The experimental setup of our previous example led to rather small multiplier spaces with $\dim\Lambda = 2048$ for the finest mesh considered. To get larger interfaces and multiplier spaces, we finally turn to the brain geometry in Figure 1. Although realistic, the geometry is still largely simplified as we have excluded the cerebellum, the aqueduct, and the central canal and expanded the subarachnoid space to allow more visible CSF pathways. Nevertheless, the geometry fully represents the complexity of the interface (gyric and sulcal brain surface), which is an important part and an additional difficulty when solving the coupled viscous-porous flow problem. Using the same discretization as before the computational mesh leads to $N_{\text{dof}} \approx 11 \cdot 10^6$ with $\dim\Lambda \approx 50 \cdot 10^3$. For the purpose of illustration we set $\mu = 3$, $K = 10^{-4}$, $D = 0.5$ and consider most of the outer surface of Ω_S with no-slip boundary condition except for a small region on the bottom where traction is prescribed. The flow field computed after 500 iterations of MinRes is plotted in Figure 4. Therein we also compare convergence of MinRes solver using preconditioner (7) with a simpler one which uses in the Λ block the operator $K(-\Delta + I)^{1/2}$, cf. the analysis in [Layton et al. 2002]. Importantly, we observe that the new preconditioner, which ignores the

intersection structure of the multiplier space, leads to very slow convergence or even divergence of the iterations. In contrast, with (7) MinRes appears to be converging. We remark that the rather slow (in comparison to Figure 3) convergence with block diagonal preconditioner (7) is related to the thin-shell geometry of the Stokes domain. In particular, the performance of block-diagonal Stokes preconditioner using the mass matrix approximation for the Schur complement is known to deteriorate for certain boundary conditions when the aspect ratio of the domain is large [Sogn and Takacs 2022].

4.3 3d-1d coupled problem

Lastly, we demonstrate how the mixed-dimensional flow problem from Section 2.3 is solved using the HAZniCS solver for metric-perturbed problems. The problem is defined by the geometry illustrated in the left subfigure of Figure 5. The neuron geometry is obtained from the NeuroMorpho.Org inventory of digitally reconstructed neurons and glia [NeuroMorpho 2017]. The neuron from a rat's brain includes a soma and 72 dendrite branches. It is embedded in a rectangular box of approximate dimensions $281\mu m \times 281\mu m \times 106\mu m$. Then, the mixed-dimensional geometry is discretized with an unstructured tetrahedral mesh in a way conforming to Γ, i.e., the $1d$ neuron mesh consists of the $3d$ edges lying on Γ. As discretization, we use $\mathbb{P}_1$ finite elements for both the $3d$ and $1d$ function spaces. Overall we end up with 641 788 degrees of freedom for the $3d$ and 3156 degrees of freedom for the $1d$ problem. Additionally, we enforce homogeneous Neumann conditions on the outer boundary of both subdomains.

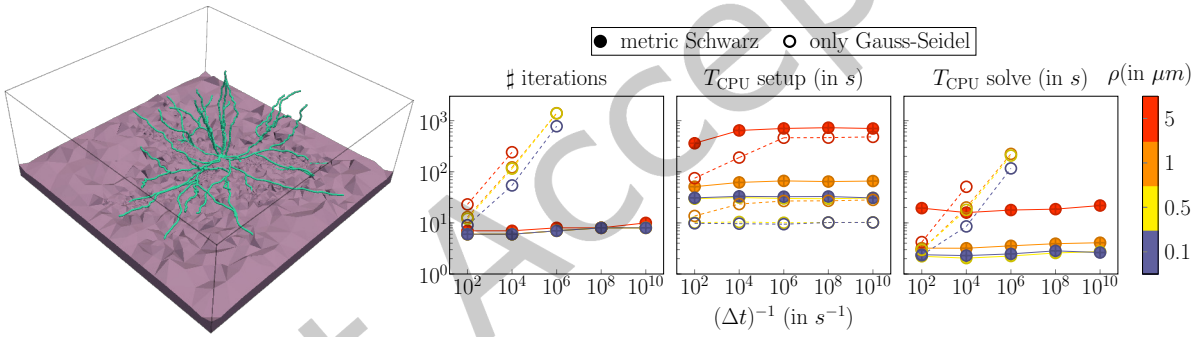


Fig. 5. (Left) Domain geometry of the $3d$-$1d$ problem (8). The $1d$ domain as the neuron and the network of neuronal dendrites is marked in cyan color and a shallow clip of the $3d$ brain tissue domain is marked in purple. The outline of the $3d$ domain is marked with black lines. (Right) Performance of the CG method to solve the system (10), preconditioned with either the metric-perturbed SA-AMG method or the standard SA-AMG with Gauss-Seidel smoother from HAZniCS. With regards to parameters ρ and Δt and up to relative residual tolerance 10^{-6}, we measure number of iterations and CPU time (T_{CPU}) required for setup and solve. Cases showing divergence and exceeding 2000 iterations are not reported. Results are obtained by running `HAZniCS-examples/demo_3d1d.py`.

To obtain the numerical solution, we use the CG method to solve the system (10) preconditioned with the metric-perturbed AMG method described in Section 3.3. The solver is executed through the call of the HAZniCS wrapper function `fenics_metric_amg_solver()` and the solver parameters are set through the input file `input.dat`. The convergence is considered reached if the l_2 relative residual norm is less than 10^{-6}. We choose the SA-AMG that uses the block Schwarz smoother (defined by the kernel decomposition (18)) for the interface degrees of freedom and the Gauss-Seidel smoother on the interior degrees of freedom. We note by the interface degrees of freedom the sub-components of the $3d$ variable that contribute to the interface current flow exchange, i.e., the nonzero components of $\Pi_\Gamma^\rho q_3$ for $q_3 \in Q_3$. The application of the block Schwarz smoothers is done in a symmetric multiplicative way, as presented in Listing 11. In addition, we compare the performance to preconditioning with

the standard SA-AMG that uses the (pointwise) Gauss-Seidel smoother in the whole system (for all degrees of freedom).

We study the performance with regards to varying the time step size Δt and the coupling/dendrite radius ρ. Specifically, we are interested in the solvers' performance for very small time steps since this results in the metric term in the system (10) to dominate. The conductivity and membrane capacitance parameters remain constant and fixed throughout their respective domains to $\sigma_3 = 3$ mS cm^{-1}, $\sigma_1 = 7$ mS cm^{-1} and $C_m = 1\,\mu$F cm^{-2} [Buccino et al. 2019]. The results are given in the collection of three subplots in Figure 5. The first subplot in Figure 5 shows stable number of iterations for the metric-perturbed AMG preconditioner, and increasing for the standard AMG and the smaller time step sizes. This is also evident in the third subplot showing required solving CPU time. In the second subplot, though, we recognize the increased setup time of the metric-perturbed AMG since it needs to construct Schwarz subspaces. Summing the total time, we can conclude that for larger time steps (miliseconds and larger) it is enough to consider the standard AMG preconditioner. However, for small time steps ($\Delta t < 10^{-2}$s) and larger coupling radii ($\rho \geq 1\mu$m), as in realistic cases, our metric-perturbed AMG method performs more efficiently and is robust with regards to problem parameters. Therefore, we can confidently incorporate the method as part of the solver for the full EMI model [Buccino et al. 2021; Jæger et al. 2021].

5 CONCLUSION

This paper introduces a collection of software solutions, HAZniCS, for solving interface-coupled multiphysics problems. The software combines two frameworks, HAZmath and FEniCS, into a flexible and powerful tool to obtain reliable and efficient simulators for various coupled problems. The focus of this work has been the (3d-2d coupled) Darcy-Stokes model and the 3d-1d coupled diffusion model for which we have presented the implementation and illustrated the performance of our solvers. In addition, we believe that the results shown in the paper demonstrate a great potential to utilize our framework in other relevant applications. The solver library allows interfacing with other finite element libraries which support the discretization of multiphysics problems, such as the new generation FEniCS platform FEniCSx or the Julia library Gridap.jl [Verdugo and Badia 2022].

ACKNOWLEDGMENTS

AB, MK, and KAM acknowledge the financial support from the Norwegian Research Council grant 300305 and 301013. The work of XH is partially supported by the National Science Foundation (NSF) under grant DMS-2208267. MK acknowledges support from Norwegian Research Council grant 303362. The research of LZ is supported in part by the U. S.-Norway Fulbright Foundation and the U. S. National Science Foundation grant DMS-2208249. The collaborative efforts of XH and LZ were supported in part by the NSF DMS-2132710 through the Workshop on Numerical Modeling with Neural Networks, Learning, and Multilevel FE.

REFERENCES

J. Adler, X. Hu, and L. Zikatanov. 2009. *HAZMATH: A Simple Finite Element, Graph, and Solver Library.* https://hazmathteam.github.io/hazmath/

V. I. Agoshkov. 1988. Poincaré-Steklov operators and domain decomposition methods in finite dimensional spaces. In *First International Symposium on Domain Decomposition Methods for Partial Differential Equations.* 73–112.

I. Ambartsumyan, E. Khattatov, I. Yotov, and P. Zunino. 2018. A Lagrange multiplier method for a Stokes–Biot fluid–poroelastic structure interaction model. *Numer. Math.* 140, 2 (2018), 513–553.

S. Badia, F. Nobile, and C. Vergara. 2009. Robin–Robin preconditioned Krylov methods for fluid–structure interaction problems. *Computer Methods in Applied Mechanics and Engineering* 198, 33-36 (2009), 2768–2784.

T. Bærland. 2019. An auxiliary space preconditioner for fractional Laplacian of negative order. *arXiv preprint arXiv:1908.04498* (2019).

T. Bærland, M. Kuchta, and K.-A. Mardal. 2019. Multigrid methods for discrete fractional Sobolev spaces. *SIAM Journal on Scientific Computing* 41, 2 (2019), A948–A972. https://doi.org/10.1137/18M1191488

S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, and et. al. 2022. *PETSc/TAO Users Manual.* Technical Report ANL-21/39 - Revision 3.18. Argonne National Laboratory.

D. M. Beazley. 1996. SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++. In *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4* (Monterey, California) *(TCLTK'96)*. USENIX Association, USA, 15. https://www.swig.org/

M. Berg, Y. Davit, M. Quintard, and S. Lorthois. 2020. Modelling solute transport in the brain microcirculation: is it really well mixed inside the blood vessels? *Journal of Fluid Mechanics* 884 (2020), A39. https://doi.org/10.1017/jfm.2019.866

R. Blaheta. 1986. A multilevel method with correction by aggregation for solving discrete elliptic problems. *Aplikace matematiky* 31, 5 (1986), 365–378.

W. M. Boon, M. Hornkjøl, M. Kuchta, K.-A. Mardal, and R. Ruiz-Baier. 2022a. Parameter-robust methods for the Biot–Stokes interfacial coupling without Lagrange multipliers. *J. Comput. Phys.* 467 (2022), 111464.

W. M. Boon, T. Koch, M. Kuchta, and K.-A. Mardal. 2022b. Robust Monolithic Solvers for the Stokes–Darcy Problem with the Darcy Equation in Primal Form. *SIAM Journal on Scientific Computing* 44, 4 (2022), B1148–B1174.

J. Bramble, J. Pasciak, and P. Vassilevski. 2000. Computational scales of Sobolev norms with application to preconditioning. *Math. Comp.* 69, 230 (2000), 463–480.

J. H. Bramble. 2019. *Multigrid methods.* Chapman and Hall/CRC.

A. Brandt, S. F. McCormick, and J. W. Ruge. 1982. *Algebraic multigrid (AMG) for automatic multigrid solutions with application to geodetic computations.* Technical Report. Inst. for Computational Studies, Fort Collins, CO.

A. P. Buccino, M. Kuchta, K. H. Jæger, T. V. Ness, P. Berthet, K.-A. Mardal, G. Cauwenberghs, and A. Tveito. 2019. How does the presence of neural probes affect extracellular potentials? *Journal of Neural Engineering* 16 (4 2019), 026030. Issue 2. https://doi.org/10.1088/1741-2552/ab03a1

A. P. Buccino, M. Kuchta, J. Schreiner, and K.-A. Mardal. 2021. *Improving Neural Simulations with the EMI Model.* Springer International Publishing, Cham, 87–98. https://doi.org/10.1007/978-3-030-61157-6_7

A. Budiša, X. Hu, M. Kuchta, K.-A. Mardal, and L. Zikatanov. 2022. *HAZniCS examples.* https://doi.org/10.5281/zenodo.7220688

A. Budiša, X. Hu, M. Kuchta, K.-A. Mardal, and L. Zikatanov. 2023a. Algebraic multigrid methods for metric-perturbed coupled problems. *arXiv preprint arXiv:2305.06073* (2023).

A. Budiša, X. Hu, M. Kuchta, K.-A. Mardal, and L. Zikatanov. 2023b. Rational Approximation Preconditioners for Multiphysics Problems. In *Numerical Methods and Applications*, Ivan Georgiev, Maria Datcheva, Krassimir Georgiev, and Geno Nikolov (Eds.). Springer Nature Switzerland, 100–113.

S. Caucao, T. Li, and I. Yotov. 2022. A multipoint stress-flux mixed finite element method for the Stokes-Biot model. *Numer. Math.* 152, 2 (2022), 411–473.

D. Cerroni, F. Laurino, and P. Zunino. 2019. Mathematical analysis, finite element approximation and numerical solvers for the interaction of 3d reservoirs with 1d wells. *GEM-International Journal on Geomathematics* 10, 1 (2019), 1–27.

S. Deparis, M. Discacciati, G. Fourestey, and A. Quarteroni. 2006. Fluid–structure algorithms based on Steklov–Poincaré operators. *Computer Methods in Applied Mechanics and Engineering* 195, 41-43 (2006), 5797–5812.

C. D'Angelo and A. Quarteroni. 2008. On the coupling of 1d and 3d diffusion-reaction equations: Application to tissue perfusion problems. *Mathematical Models and Methods in Applied Sciences* 18, 8 (2008), 1481–1504.

P. K. Eide, V. Vinje, A. H. Pripp, K.-A. Mardal, and G. Ringstad. 2021. Sleep deprivation impairs molecular clearance from the human brain. *Brain* 144, 3 (2021), 863–874.

R. D. Falgout and U. M. Yang. 2002. hypre: A Library of High Performance Preconditioners. In *Computational Science — ICCS 2002*, Peter M. A. Sloot, Alfons G. Hoekstra, C. J. Kenneth Tan, and Jack J. Dongarra (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 632–641.

T. Führer. 2022. Multilevel decompositions and norms for negative order Sobolev spaces. *Math. Comp.* 91, 333 (2022), 183–218.

J. Galvis and M. Sarkis. 2007. Non-matching mortar discretization analysis for the coupling Stokes-Darcy equations. *Electronic Transactions on Numerical Analysis* 26 (2007), 350–384.

I. G. Gjerde, K. Kumar, and J. M. Nordbotten. 2020. A singularity removal method for coupled 1D–3D flow models. *Computational Geosciences* 24, 2 (2020), 443–457.

S. Harizanov, R. Lazarov, S. Margenov, and P. Marinov. 2020. Numerical solution of fractional diffusion–reaction problems based on BURA. *Computers & Mathematics with Applications* 80, 2 (2020), 316–331.

S. Harizanov, R. Lazarov, S. Margenov, P. Marinov, and Y. Vutov. 2018. Optimal solvers for linear systems with fractional powers of sparse SPD matrices. *Numerical Linear Algebra with Applications* 25, 5 (2018), e2167.

S. Harizanov, I. Lirkov, and S. Margenov. 2022. Rational Approximations in Robust Preconditioning of Multiphysics Problems. *Mathematics* 10, 5 (2022), 780.

G. Hartung, S. Badr, S. Mihelic, A. Dunn, X. Cheng, S. Kura, D. A. Boas, D. Kleinfeld, A. Alaraj, and A. A. Linninger. 2021. Mathematical synthesis of the cortical circulation for the whole mouse brain—part II: Microcirculatory closure. *Microcirculation* 28, 5 (2021), e12687.

R. Hiptmair and J. Xu. 2007. Nodal Auxiliary Space Preconditioning in H(curl) and H(div) Spaces. *SIAM J. Numer. Anal.* 45, 6 (2007), 2483–2509. https://doi.org/10.1137/060660588

C. Hofreither. 2020. A unified view of some numerical methods for fractional diffusion. *Computers and Mathematics with Applications* 80, 2 (2020), 332–350. https://doi.org/10.1016/j.camwa.2019.07.025

K.-E. Holter, M. Kuchta, and K.-A. Mardal. 2020. Robust preconditioning of monolithically coupled multiphysics problems. *arXiv preprint arXiv:2001.05527* (2020).

K. E. Holter, M. Kuchta, and K.-A. Mardal. 2021. Robust preconditioning for coupled Stokes–Darcy problems with the Darcy problem in primal form. *Computers & Mathematics with Applications* 91 (2021), 53–66.

M. Hornkjøl, L. M. Valnes, G. Ringstad, M. E. Rognes, P.-K. Eide, K.-A. Mardal, and V. Vinje. 2022. CSF circulation and dispersion yield rapid clearance from intracranial compartments. *Frontiers in Bioengineering and Biotechnology* 10 (2022), 932469.

X. Hu, J. Lin, and L. T. Zikatanov. 2019. An Adaptive Multigrid Method Based on Path Cover. *SIAM Journal on Scientific Computing* 41, 5 (Jan. 2019), S220–S241. https://doi.org/10.1137/18M1194493 Citation Key Alias: HuLinZikatanov2018.

X. Hu, P. S. Vassilevski, and J. Xu. 2016. A two-grid SA-AMG convergence bound that improves when increasing the polynomial degree: Improving TG convergence with increasing smoothing steps. *Numerical Linear Algebra with Applications* 23, 4 (Aug. 2016), 746–771. https://doi.org/10.1002/nla.2053

X. Hu, S. Wu, X.-H. Wu, J. Xu, C.-S. Zhang, S. Zhang, and L. Zikatanov. 2013. Combined preconditioning with applications in reservoir simulation. *Multiscale Modeling & Simulation* 11, 2 (2013), 507–521.

J. J. Iliff, M. Wang, Y. Liao, B. A. Plogg, W. Peng, G. A. Gundersen, H. Benveniste, G. E. Vates, R. Deane, S. A. Goldman, E. A. Nagelhus, and M. Nedergaard. 2012. A Paravascular Pathway Facilitates CSF Flow Through the Brain Parenchyma and the Clearance of Interstitial Solutes, Including Amyloid β. *Science Translational Medicine* 4, 147 (2012), 147ra111.

K. H. Jæger, K. G. Hustad, X. Cai, and A. Tveito. 2021. *Operator Splitting and Finite Difference Schemes for Solving the EMI Model.* Springer International Publishing, Cham, 44–55. https://doi.org/10.1007/978-3-030-61157-6_4

R. T. Kedarasetti, K. L. Turner, C. Echagarruga, B. J. Gluckman, P. J. Drew, and F. Constanzo. 2020. Functional hyperemia drives fluid exchange in the paravascular space. *Fluids Barriers CNS* 17, 52 (2020). https://doi.org/10.1186/s12987-020-00214-3

D. H. Kelley, T. Bohr, P. G. Hjorth, S. C. Holst, S. Hrabětová, V. Kiviniemi, T. Lilius, I. Lundgaard, K.-A. Mardal, E. A. Martens, et al. 2022. The glymphatic system: Current understanding and modeling. *Iscience* (2022).

H. Kim, J. Xu, and L. Zikatanov. 2003. A multigrid method based on graph matching for convection diffusion equations. *Numerical linear algebra with applications* (2003). http://onlinelibrary.wiley.com/doi/10.1002/nla.317/abstract

T. Koch, M. Schneider, R. Helmig, and P. Jenny. 2020. Modeling tissue perfusion in terms of 1d-3d embedded mixed-dimension coupled problems with distributed sources. *J. Comput. Phys.* 410 (2020). https://doi.org/10.1016/j.jcp.2020.109370

T. V. Kolev and P. S. Vassilevski. 2012. Parallel auxiliary space amg solver for H(div) problems. *SIAM Journal on Scientific Computing* 34 (2012), 1–21. Issue 6. https://doi.org/10.1137/110859361

T. Köppl, E. Vidotto, B. Wohlmuth, and P. Zunino. 2018. Mathematical modeling, analysis and numerical approximation of second-order elliptic problems with inclusions. *Mathematical Models and Methods in Applied Sciences* 28, 5 (2018), 953–978.

M. Kuchta. 2021. Assembly of Multiscale Linear PDE Operators. In *Numerical Mathematics and Advanced Applications ENUMATH 2019*, Fred J. Vermolen and Cornelis Vuik (Eds.). Springer International Publishing, Cham, 641–650. https://doi.org/10.1007/978-3-030-55874-1_63

M. Kuchta, F. Laurino, K. A. Mardal, and P. Zunino. 2021. Analysis and approximation of mixed-dimensional PDEs on 3D-1D domains coupled with Lagrange multipliers. *SIAM J. Numer. Anal.* 59, 1 (2021), 558–582. https://doi.org/10.1137/20M1329664

F. Laurino and P. Zunino. 2019. Derivation and analysis of coupled PDEs on manifolds with high dimensionality gap arising from topological model reduction. *ESAIM: Mathematical Modelling and Numerical Analysis* 53, 6 (2019), 2047–2080.

W. J. Layton, F. Schieweck, and I. Yotov. 2002. Coupling fluid flow with porous media flow. *SIAM J. Numer. Anal.* 40, 6 (2002), 2195–2218.

Y. Lee, J. Wu, J. Xu, and L. T. Zikatanov. 2007. Robust subspace correction methods for nearly singular systems. *Mathematical Models and Methods in Applied Sciences* 17, 11 (2007), 1937–1963.

T. Li and I. Yotov. 2022. A mixed elasticity formulation for fluid–poroelastic structure interaction. *ESAIM: Mathematical Modelling and Numerical Analysis* 56, 1 (2022), 1–40.

A. Logg, K.-A. Mardal, G. N. Wells, et al. 2012. *Automated Solution of Differential Equations by the Finite Element Method.* Springer, Berlin Heidelberg. https://doi.org/10.1007/978-3-642-23099-8

K.-A. Mardal and J. B. Haga. 2012. Block preconditioning of systems of PDEs. In *Automated solution of differential equations by the finite element method.* Springer, Berlin Heidelberg, 643–655. https://doi.org/10.1007/978-3-642-23099-8_35

K.-A. Mardal, M. E. Rognes, T. B. Thompson, and L. M. Valnes. 2022. *Mathematical modeling of the human brain: from magnetic resonance images to finite element simulation.* Springer, Berlin Heidelberg.

K.-A. Mardal and R. Winther. 2011. Preconditioning discretizations of systems of partial differential equations. *Numerical Linear Algebra with Applications* 18, 1 (2011), 1–40. https://doi.org/10.1002/nla.716

I. Marek. 1991. Aggregation methods of computing stationary distributions of Markov processes. In *Numerical Treatment of Eigenvalue Problems Vol. 5/Numerische Behandlung von Eigenwertaufgaben Band 5.* Springer, 155–169.

S. Míka and P. Vaněk. 1992. Acceleration of convergence of a two level algebraic algorithm by aggregation in smoothing process. *Appl. Math.* 37 (1992), 343–356.

S. Míka and P. Vaněk. 1992. A Modification of the two-level algorithm with overcorrection. *Appl. Math.* 37 (1992), 13–28.

Y. Nakatsukasa, O. Sète, and L. N. Trefethen. 2018. The AAA Algorithm for Rational Approximation. *SIAM Journal on Scientific Computing* 40, 3 (2018), A1494–A1522. https://doi.org/10.1137/16M1106122

NeuroMorpho 2017. *Digital reconstruction of a neuron, ID NMO_72183.* https://neuromorpho.org/neuron_info.jsp?neuron_name=P14_rat1_layerIII_cell1

A. Quarteroni and A. Valli. 1991. Theory and application of Steklov-Poincaré operators for boundary-value problems. In *Applied and Industrial Mathematics*. Springer, 179–203.

B. Rivière and I. Yotov. 2005. Locally conservative coupling of Stokes and Darcy flows. *SIAM J. Numer. Anal.* 42, 5 (2005), 1959–1977.

J. Sogn and S. Takacs. 2022. Stable discretizations and IETI-DP solvers for the Stokes system in multi-patch Isogeometric Analysis. *arXiv preprint arXiv:2202.13707* (2022).

A. Tveito, K. H. Jæger, M. Kuchta, K.-A. Mardal, and M. E. Rognes. 2017. A cell-based framework for numerical modeling of electrical conduction in cardiac tissue. *Frontiers in Physics* (2017), 48.

I. Vakhutinsky, L. Dudkin, and A. Ryvkin. 1979. Iterative aggregation–A new approach to the solution of large-scale problems. *Econometrica: Journal of the Econometric Society* (1979), 821–841.

P. Vaněk, J. Mandel, and M. Brezina. 1996. Algebraic multigrid based on smoothed aggregation for second and fourth order problems. *Computing* 56 (1996), 179–196.

P. Vaněk, J. Mandel, and M. Brezina. 1998. Convergence of algebraic multigrid based on smoothed aggregation. *Computing* 56 (1998), 179–196.

F. Verdugo and S. Badia. 2022. The software design of Gridap: A Finite Element package based on the Julia JIT compiler. *Computer Physics Communications* 276 (2022), 108341.

L. Xie, H. Kang, Q. Xu, M. J. Chen, Y. Liao, M. Thiyagarajan, J. O'Donnell, D. J. Christensen, C. Nicholson, J. J. , et al. 2013. Sleep drives metabolite clearance from the adult brain. *science* 342, 6156 (2013), 373–377.

J. Xu. 1992. Iterative methods by space decomposition and subspace correction. *SIAM Rev.* 34, 4 (1992), 581–613. https://doi.org/10.1137/1034116

J. Xu and L. Zikatanov. 2002. The method of alternating projections and the method of subspace corrections in Hilbert space. *J. Amer. Math. Soc.* 15, 3 (2002), 573–597. https://doi.org/10.1090/S0894-0347-02-00398-3

J. Xu and L. Zikatanov. 2017. Algebraic multigrid methods. *Acta Numer.* 26 (2017), 591–721. https://doi.org/10.1017/S0962492917000083

X. Zhao, X. Hu, W. Cai, and G. E. Karniadakis. 2017. Adaptive finite element method for fractional differential equations using hierarchical matrices. *Computer Methods in Applied Mechanics and Engineering* 325 (Oct. 2017), 56–76.