Electronic Transactions on Numerical Analysis.
Volume 66, pp. 108–132, 2026.

ISSN 1068–9613.
DOI: 10.1553/etna_vol66s108

ETNA
Kent State University and
Johann Radon Institute (RICAM)

UNSTRUCTURED TO STRUCTURED: GEOMETRIC MULTIGRID ON COMPLEX GEOMETRIES VIA DOMAIN REMAPPING*

NICOLAS NYTKO†, SCOTT MACLACHLAN‡, J. DAVID MOULTON§, LUKE N. OLSON¶, ANDREW REISNER‖, AND MATTHEW WEST**

Abstract. For domains that are easily represented by structured meshes, robust geometric multigrid solvers can quickly provide the numerical solution to many discretized elliptic PDEs. However, for complicated domains with unstructured meshes, constructing suitable hierarchies of meshes becomes challenging. We propose a framework for mapping computations from such complex domains to regular computational domains via diffeomorphisms, enabling the use of robust geometric-style multigrid. This mapping facilitates regular memory accesses during solves, improving efficiency and scalability, especially on massively parallel processors such as GPUs. Moreover, we show that the diffeomorphic mapping itself may be approximately learned using an invertible neural network, facilitating automated application to geometries where no analytic mapping is readily available.

Key words. geometric multigrid, diffeomorphic, LDDMM, neural ODE, structured grid

AMS subject classifications. 65L60, 65N55

1. Introduction.

Geometric multigrid methods are efficient iterative solvers for elliptic partial differential equations (PDEs), offering an optimal time complexity and rapid convergence for problems where geometric structure can be exploited. However, extending multigrid to arbitrary meshes—finite partitions of domains, which are open subsets of $\mathbb{R}^d$—remains challenging due to the difficulty of constructing a suitable hierarchy of coarse meshes that maintains multigrid efficiency. In practice, algebraic multigrid (AMG) methods [30, 32] are often used on complex meshes as they do not require geometric information about the problem, operating directly on the linear system itself. Despite their versatility, AMG approaches tend to suffer from higher setup costs, as more expensive graph algorithms are required to coarsen the problem. Moreover, AMG can suffer from poor data locality and irregular data accesses due to sparse-matrix storage patterns that hinder performance on massively parallel architectures such as GPUs [2].

To address these limitations, we propose a novel framework to expand the applicability of geometric multigrid methods to a larger class of problems. To do so, we construct diffeomorphic mappings between a given mesh and a structured (regular) mesh on which robust geometric multigrid methods can be used. A full multigrid hierarchy is then constructed by transferring solution and/or residual vectors from the given (unstructured) mesh to the structured one and employing a black-box multigrid solver [10, 12, 29] to quickly and efficiently solve the structured problem. This auxiliary mesh correction or solution is then transferred back to the unstructured mesh, where only cheap, local relaxation is performed. This results in a solver that leverages the regularity of the structured problem to yield an efficient solver, while also allowing for essential features of the physical geometry to be preserved.

*Received August 15, 2025. Accepted July 24, 2026. Published online on September 11, 2026. Recommended by Stefan Vandewalle.
†Corresponding author. Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign (nnytko2@illinois.edu).
‡Mathematics and Statistics, Memorial University of Newfoundland.
§Energy and Natural Resources Security (EES-16), Los Alamos National Laboratory.
¶Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign.
‖Computing and Artificial Intelligence (CAI-1), Los Alamos National Laboratory.
**Mechanical Sciences and Engineering, University of Illinois Urbana-Champaign.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Additionally, we show that the diffeomorphic mapping between domains can be learned. Taking inspiration from existing work on large-deformation diffeomorphic metric mappings (LDDMMs) [20], used heavily in computational anatomy, we demonstrate that such a mapping can be approximately learned for geometries lacking an analytic conformal mapping. We show numerical results for both domains with existing analytic mappings and ones where the mapping itself is learned. Iteration counts to convergence and time to solution of the resulting geometric solver are compared to those of algebraic methods, showing that our framework is relatively robust and comparable with other multilevel solvers.

A variety of related works have explored the usage of structured discretizations for numerical PDE problems posed on complex geometries, using the structure both to outright solve the PDE and also to form a nested multigrid hierarchy for efficient computation. Cut finite-element methods (CutFEM) [6] overlay a structured background mesh over a geometric description of a domain, weakly enforcing boundary and interface conditions through Nitsche's method [22] and additional penalty terms to enforce regularity. Overlapping grid methods [9, 17, 18] approximate a complex "Chimera" domain by combining multiple overset structured grids with simple (shear) transforms applied. There also exist works studying projection-based interpolation of functions between nonmatching overlapping finite-element spaces [21, 25], optionally imposing restrictions such as weak conservation of mass or preserving boundary norms. This is, in effect, similar to how we compute the projection operator for our multilevel solver. In multigrid contexts, the auxiliary space method [5, 7, 33] constructs a structured auxiliary problem and stable transfer operators to precondition unstructured (fine-level) discretizations. In contrast, our framework builds this auxiliary space through smooth geometric mappings. Recently in the scientific machine-learning domain, diffeomorphic transformations have been similarly used to solve PDEs on regular domains by use of Fourier neural operators (FNOs) [23, 24]; similarly to this work, both analytic and learned mappings are considered.

Throughout this work, we will explicitly use the term *domain* to refer to a connected open subset of $\mathbb{R}^d$, and *mesh* to refer to its respective finite partition composed of simplices or tensor-product elements. This distinction helps distinguish between the continuous geometric setting and the discrete numerical approximation.

The remainder of this paper is outlined as follows. We introduce our method in Section 2, including details on the multigrid method in Section 2.2 and the grid-transfer operators between the structured and unstructured grids in Sections 2.3 and 2.4. Details of the construction and learning of mappings between structured and unstructured grids is presented in Section 3. Numerical results are given in Section 4, with conclusions given in Section 5.

2. Method. We consider the elliptic diffusion problem with Dirichlet boundary conditions,

$$\begin{aligned} -\nabla \cdot \mathcal{D}\nabla u &= f \quad \text{in } \Omega_p, \\ u &= g \quad \text{on } \partial\Omega_p, \end{aligned} \tag{2.1}$$

defined over an open, connected space (the "physical domain") $\Omega_p \subset \mathbb{R}^d$ with continuous boundary $\partial\Omega_p$ and symmetric, positive-definite diffusion tensor $\mathcal{D} : \mathbb{R}^d \to \mathbb{R}^{d\times d}$. We assume there exists a triangulation $\mathcal{T}_p = \{\tau_p^i\}$ such that $\Omega_p = \bigcup_i \tau_p^i$, although we will demonstrate below that our approach naturally generalizes to the case where $\Omega_p \approx \bigcup_i \tau_p^i$, such as when the physical domain has curved boundaries. In this work, we consider the case of $d = 2$, though the method directly extends to $d = 3$ as well. Similarly, we focus on the diffusion equation (2.1), although the method could also be applied to reaction–diffusion or convection–diffusion equations.

In addition to the physical domain, Ω_p, we define an auxiliary *computational domain*, $\Omega_c = (-1, 1)^d$ and, as discussed below, a corresponding triangulation, $\mathcal{T}_c = \{\tau_c^i\}$, with

ETNA
Kent State University and
Johann Radon Institute (RICAM)

$\Omega_c = \bigcup_i \tau_c^i$. As result, both Ω_p and Ω_c are smooth manifolds, allowing us to additionally define an invertible map $T : \overline{\Omega}_p \mapsto \overline{\Omega}_c$ between the two domains, including their boundaries. Moreover, we require T to be smooth (diffeomorphic) over the interior to ensure the Jacobian $\boldsymbol{J}_T(\boldsymbol{x})$ and its inverse exist and are continuous. We use $|\cdot|$ to denote the absolute value of the determinant when applied to a matrix; for example, $|\boldsymbol{J}_T| = |\det(\boldsymbol{J}_T)|$. In what follows, we will assume $\mathcal{T}_c$ corresponds to a logically structured grid of regular elements, such as a tensor-product mesh of quadrilateral elements. A schematic of the domains and meshes is shown in Figure 2.1. We first consider example domains where T is known analytically. In Section 3.1, however, we will show that T can also be learned for more complex domains.

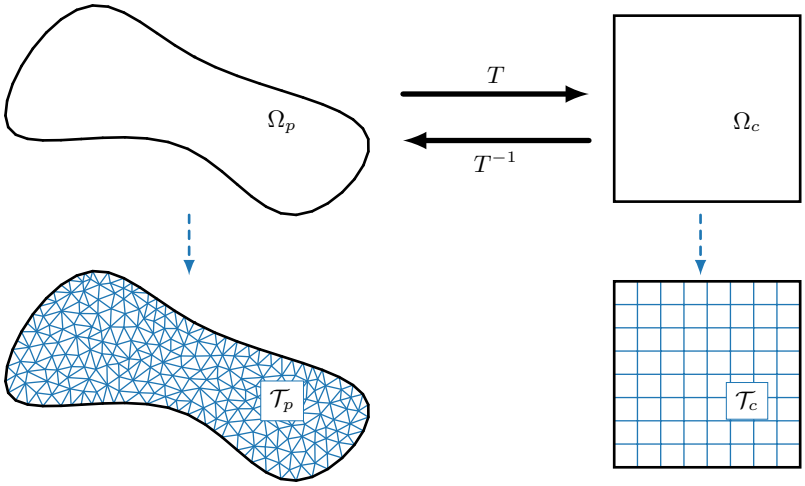


FIG. 2.1. *Schematic of the* physical *domain (upper left), the* computational *domain (upper right), and their respective meshes, $\mathcal{T}_p$ and $\mathcal{T}_c$. The mapping $T : \Omega_p \to \Omega_c$ is assumed invertible.*

Our method assumes knowledge both of the continuum PDE, given in equation (2.1), and of the mapping, T. As shown below, this allows us to define an equivalent weak form of the PDE over the computational domain. We make use of this to define a structured-multigrid hierarchy (using robust geometric multigrid components) on which most of the computational work in a multigrid V-cycle is performed. It is well known that structured-grid methods are generally the most efficient and straightforward of the multigrid methods, due to their highly regular memory access patterns which do not need the indirection typical of generic sparse matrix formats, such as the compressed sparse row (CSR) format. This clearly has high potential to yield speedups on compute platforms, such as GPUs, that have performance bottlenecks resulting from memory transfers, rather than raw compute power.

We begin with the regular weak form for the diffusion equation on the physical domain, Ω_p, where we write the solution to equation (2.1) as $u(\boldsymbol{x}) = \hat{u}_p(\boldsymbol{x}) + \hat{g}(\boldsymbol{x})$, where $\hat{g} \in H^1(\Omega_p)$ satisfies the boundary condition $\hat{g}(\boldsymbol{x}) = g$ on $\partial\Omega_p$, and we seek the function $\hat{u}_p \in H_0^1(\Omega_p)$ that satisfies

$$\int_{\Omega_p} \mathcal{D}\,\nabla\hat{u}_p \cdot \nabla v \,\mathrm{d}x = \int_{\Omega_p} f v \,\mathrm{d}x - \int_{\Omega_p} \mathcal{D}\,\nabla\hat{g} \cdot \nabla v \,\mathrm{d}x \quad \forall\, v \in H_0^1(\Omega_p), \tag{2.2}$$

where $H^1(\Omega_p)$ is the standard Sobolev space of square-integrable functions over Ω_p with square-integrable first derivatives, and $H_0^1(\Omega_p)$ is the restriction of $H^1(\Omega_p)$ to functions that vanish on the boundary. The weak form can be mapped to the computational domain by the

ETNA
Kent State University and
Johann Radon Institute (RICAM)

change of coordinates defined by T, in a similar fashion to standard finite-element assembly on a general, triangular mesh. We transform the weak form in equation (2.2) to one over Ω_c using $\boldsymbol{J}_T$, asking to find $\hat{u}_c \in H_0^1(\Omega_c)$ such that

$$\begin{aligned}(2.3)\qquad &\int_{\Omega_c} ((\mathcal{D}\circ T^{-1})\boldsymbol{J}_T^T\nabla\hat{u}_c)\cdot(\boldsymbol{J}_T^T\nabla v)\,|\boldsymbol{J}_T^{-1}|\,\mathrm{d}x \\ &\quad= \int_{\Omega_c}(f\circ T^{-1})v\,|\boldsymbol{J}_T^{-1}|\,\mathrm{d}x \\ &\quad\quad- \int_{\Omega_c}((\mathcal{D}\circ T^{-1})\boldsymbol{J}_T^T\nabla(\hat{g}\circ T^{-1}))\cdot(\boldsymbol{J}_T^T\nabla v)|\boldsymbol{J}_T^{-1}|\,\mathrm{d}x \quad \forall\, v\in H_0^1(\Omega_c).\end{aligned}$$

We note that we write $\hat{u}_p$ for the solution in the physical domain and $\hat{u}_c$ for the solution in the computational domain, which are related by composition with T or T^{-1}, while we explicitly compose the right-hand side data, f and $\hat{g}$ (defined on the physical domain), with $T^{-1}:\Omega_c\to\Omega_p$ when writing the weak form in the computational domain. In practice, we will not need to evaluate the right-hand side in equation (2.3), so we need not consider the details of these mappings.

2.1. Discretization. We discretize the weak forms over the physical and computational domains, equations (2.2) and (2.3), by restricting the test and trial function spaces to discrete function spaces $\mathcal{V}_p\subset H_0^1(\Omega_p)$ and $\mathcal{V}_c\subset H_0^1(\Omega_c)$. As is natural with finite-element methods, these function spaces are associated with the meshes, $\mathcal{T}_p$ and $\mathcal{T}_c$, although we note that both the meshes and the discretization spaces can be chosen completely independently, as depicted in Figure 2.1. For both $\mathcal{V}_p$ and $\mathcal{V}_c$, we assume there exist known bases such that any function, $f_p\in\mathcal{V}_p$, can be written as a weighted sum of finitely many basis functions, $f_p=\sum_i^{N_p} f_p^i\phi_p^i$, with a similar expression for any $f_c\in\mathcal{V}_c$. We then write $\boldsymbol{f}_p\in\mathbb{R}^{N_p}$ as the vector whose entries are the coefficients $\{f_p^i\}$.

While the framework we present is general enough for any discrete bases to be used for the physical and computational spaces, in this work we consider the case where $\mathcal{V}_p=P_1(\mathcal{T}_p)$, piecewise linear functions on a triangular mesh of the physical domain, and $\mathcal{V}_c=Q_1(\mathcal{T}_c)$, piecewise bilinear functions on a tensor-product quadrilateral mesh of the computational domain. The integrals in the bilinear forms from equations (2.2) and (2.3) are numerically approximated and assembled into matrices, $\boldsymbol{A}_p$ and $\boldsymbol{A}_c$, for the physical and computational operators, respectively. This integration and assembly step is done as standard for finite elements; in practice, the order required of the quadrature rule depends on the smoothness of $\mathcal{D}$ and the regularity of the mapping T, with rapidly changing PDE coefficients and/or ill-conditioned maps (with rapid oscillations/changes) requiring higher-order quadrature schemes. Thus, when constructing maps to the computational domain, we aim to compute maps that are as smooth as possible, to minimize the induced computational cost of the mapping.

An important consequence of the mapping to the computational domain is that we must solve a different problem there than on the physical domain, due to anisotropy induced by the mapping. One way to quantify the modified anisotropy, and the difficulty it may introduce, is to consider the effective diffusion coefficient, $\mathcal{D}_c$, on the computational domain, given by

$$(2.4)\qquad \mathcal{D}_c=\boldsymbol{J}_T(\mathcal{D}\circ T^{-1})\boldsymbol{J}_T^T|\boldsymbol{J}_T^{-1}|.$$

First of all, we note that there is a natural (but not sharp) bound on the condition number of $\mathcal{D}_c$, $\mathrm{cond}(\mathcal{D}_c)$, as

$$\mathrm{cond}(\mathcal{D}_c(\hat{\boldsymbol{x}}))\le\mathrm{cond}(\boldsymbol{J}_T)^2\,\mathrm{cond}(\mathcal{D}(\boldsymbol{x})),$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

when $\hat{\boldsymbol{x}} = T^{-1}(\boldsymbol{x})$ for some $\boldsymbol{x} \in \Omega_p$. Thus, $\mathcal{D}_c$ reflects any anisotropy present in $\mathcal{D}$, while ill-conditioning present in T can be amplified quadratically in the resulting diffusivity coefficient. This can be either helpful or harmful, depending on whether these align with those of $\mathcal{D}$ and, if they do, whether they cancel out ill-conditioning in $\mathcal{D}$ or compound it. Secondly, we note that rewriting the diffusivity coefficient in this form allows easy use of existing finite-element discretizations on structured grids, without needing special treatment of the weak form in equation (2.3).

In most cases, the added anisotropy causes the problem on the computational domain to be *more difficult*. However, as an example where the diffeomorphic transformation *improves* the conditioning of $\mathcal{D}$, consider the physical domain given by the stretched rectangle, $\Omega_p = (0, 10) \times (0, 1)$, with constant diffusivity $\mathcal{D} = \mathrm{diag}(1, 10^{-2})$. As a result, the material is more diffusive along the horizontal axis. The map T to the computational domain is given by the affine map

$$T(\boldsymbol{x}) = \begin{bmatrix} 2/10 & \\ & 2 \end{bmatrix} \boldsymbol{x} - \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{2.5}$$

Taking the Jacobian as the linear part of equation (2.5) and substituting into equation (2.4), we obtain an effective diffusion over the computational domain given as

$$\mathcal{D}_c = \frac{5}{2} \begin{bmatrix} 1/5 & \\ & 2 \end{bmatrix} \begin{bmatrix} 1 & \\ & 1/100 \end{bmatrix} \begin{bmatrix} 1/5 & \\ & 2 \end{bmatrix} = \frac{1}{10} \boldsymbol{I},$$

which improves the conditioning of the diffusivity tensor from $\mathrm{cond}(\mathcal{D}) = 100$ to $\mathrm{cond}(\mathcal{D}_c) = 1$.

2.2. Multigrid solver. Our solution methodology makes use of the following ingredients:

1. the linear system for the PDE on the physical domain, $\boldsymbol{A}_p \boldsymbol{u}_p = \boldsymbol{f}_p$, defined either through an explicit matrix, $\boldsymbol{A}_p$, or by way of a method to compute matrix–vector products;
2. a stationary iteration, S, to (cheaply) relax residuals of $\boldsymbol{A}_p \boldsymbol{u}_p = \boldsymbol{f}_p$;
3. matrix $\boldsymbol{A}_c$ corresponding to the bilinear form on the left-hand side of equation (2.3); and
4. an interpolation operator, $\boldsymbol{P}$, mapping from $\mathcal{V}_c$ to $\mathcal{V}_p$ as described below in Section 2.3.

We solve the physical system, $\boldsymbol{A}_p \boldsymbol{u}_p = \boldsymbol{f}_p$, using a two-grid multigrid method, with the computational grid playing the role of the coarse-grid operator, and $\boldsymbol{P}$ and $\boldsymbol{P}^T$ as the grid-transfer operators. We solve the coarse-grid system, with matrix $\boldsymbol{A}_c$, by applying the black-box multigrid (BoxMG) algorithm [10, 12] as implemented in the Cedar software package [29], though any structured-grid multigrid solver may be used.

Cedar provides a flexible and modern C++ control layer and interface to legacy BoxMG Fortran kernels, and hence it uses structured coarsening (by a factor of two in each direction) with operator-induced interpolation and a Galerkin coarse-grid operator to provide a robust solver for problems with discontinuous coefficients. In addition, it provides various point and line relaxation [28] schemes to efficiently solve both isotropic and anisotropic problems. However, for consistency we use only pointwise relaxation in our results.

The computational grid is coarsened using the structured coarsening present in BoxMG and Cedar to obtain a computational hierarchy. Thus, the full hierarchy in our approach is formed by prepending the original physical system to the computational hierarchy, as shown in Figure 2.2. We present details of the solver algorithm in the following sections.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

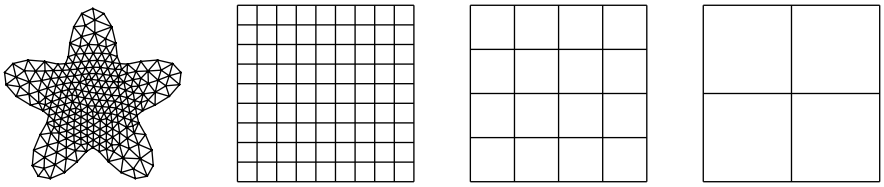


FIG. 2.2. *An example multigrid hierarchy. The finest level (left-most mesh) corresponds to the mesh for the physical domain, followed by the hierarchy of structured grids (right three meshes) on the computational domain.*

2.2.1. Setup phase. The setup phase of the algorithm is primarily that of BoxMG on the matrix $\boldsymbol{A}_c$, but also includes setting up the transfers between the finest computational grid and the physical grid.

As inputs, the setup phase requires knowledge of the physical and computational meshes, $\mathcal{T}_p$ and $\mathcal{T}_c$, as well as the mapping, T. Either $\boldsymbol{A}_c$ must be provided, or the diffusion coefficient, $\mathcal{D}$, must be known. With this, the setup phase of the algorithm requires the following:

1. Compute interpolation $\boldsymbol{P}$ by mapping vertices of $\mathcal{T}_p$ via T (see Section 2.3).
2. If needed, compute $\boldsymbol{A}_c$ from basis for $\mathcal{V}_c$ and equation (2.3).
3. Setup BoxMG solver on $\boldsymbol{A}_c$.

2.2.2. Solution phase. The solution phase of the algorithm consists of stationary iterations wrapped around a V-cycle multigrid structure. This requires the physical grid approximate solution, $\boldsymbol{u}_p$, right-hand side, $\boldsymbol{f}_p$, and matrix, $\boldsymbol{A}_p$, as well as the relaxation operator, S, and the computational grid BoxMG solver. A single V-cycle then takes essentially the standard form:

1. Pre-relaxation: $\boldsymbol{u}_p \leftarrow S(\boldsymbol{f}_p - \boldsymbol{A}_p\boldsymbol{u}_p)$.
2. Restrict residual: $\boldsymbol{r}_c \leftarrow \boldsymbol{P}^T(\boldsymbol{f}_p - \boldsymbol{A}_p\boldsymbol{u}_p)$.
3. BoxMG V-cycle on $\boldsymbol{A}_c\boldsymbol{e}_c = \boldsymbol{r}_c$.
4. Prolong correction: $\boldsymbol{u}_p \leftarrow \boldsymbol{u}_p + \boldsymbol{P}\boldsymbol{e}_c$.
5. Post-relaxation: $\boldsymbol{u}_p \leftarrow S(\boldsymbol{f}_p - \boldsymbol{A}_p\boldsymbol{u}_p)$.

2.3. Interpolation. Clearly, the success or failure of the proposed method depends strongly on the quality of the grid-transfer operator, $\boldsymbol{P}$, from Ω_c to Ω_p. We define this based on the minimization problem: given a function, $u_c \in \mathcal{V}_c$, we want to find the *nearest* (in an L^2 sense) function $u_p \in \mathcal{V}_p$ under the map T,

$$u_p = \operatorname*{argmin}_{u_p \in \mathcal{V}_p} \frac{1}{2}\|u_p - u_c \circ T\|^2_{L^2(\Omega_p)}. \tag{2.6}$$

This is minimized by solving the associated Euler–Lagrange equation [16], leading to the equality

$$\int_{\Omega_p} u_p v \,\mathrm{d}x = \int_{\Omega_p} (u_c \circ T) v \,\mathrm{d}x \quad \forall\, v \in \mathcal{V}_p,$$

or, equivalently, when u_c and u_p are expanded out in terms of their respective basis functions,

$$\sum_{j=1}^{N_p} u_p^j \int_{\Omega_p} \phi_p^j \phi_p^i \,\mathrm{d}x = \sum_{j=1}^{N_c} u_c^j \int_{\Omega_p} (\phi_c^j \circ T)\phi_p^i \,\mathrm{d}x \quad \forall\, \phi_p^i \in \mathcal{V}_p.$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Since the equation is linear in the coefficients u_p^j and u_c^j, we can express the above equality as

$$\boldsymbol{M}_p\boldsymbol{u}_p = \boldsymbol{C}\boldsymbol{u}_c \quad \Longrightarrow \quad \boldsymbol{u}_p = \boldsymbol{M}_p^{-1}\boldsymbol{C}\boldsymbol{u}_c,$$

where the bilinear *physical* and *coupled* mass forms, respectively, are defined by

$$\begin{aligned}[\boldsymbol{M}_p]_{ij} &= \int_{\Omega_p} \phi_p^j \phi_p^i \,\mathrm{d}x,\\ [\boldsymbol{C}]_{ij} &= \int_{\Omega_p} (\phi_c^j \circ T)\phi_p^i \,\mathrm{d}x.\end{aligned} \tag{2.7}$$

As $\boldsymbol{u}_c$ is an arbitrary function in $\mathcal{V}_c$, the general prolongation operator becomes $\boldsymbol{P} = \boldsymbol{M}_p^{-1}\boldsymbol{C}$. In practice, when we apply $\boldsymbol{P}$ to a vector, we only approximately invert the mass matrix, using just two iterations of the unpreconditioned conjugate gradient method (CG).

When seeking a restriction from the physical to the computational space, we aim to find the functional $u_c^* \in \mathcal{V}_c^*$ that is nearest to a given $u_p^* \in \mathcal{V}_p^*$. An important observation is that we are seeking elements in the dual of the function space; in a multigrid setting, residuals are defined by a functional corresponding to evaluations of the bilinear form and the right-hand side.

By the Riesz representation theorem, given functionals u_c^* and u_p^*, we have unique $\hat{u}_c$ and $\hat{u}_p$ such that

$$\begin{aligned}u_c^*(v) &= \int_{\Omega_c} \hat{u}_c v \,\mathrm{d}x \quad \forall\, v \in \mathcal{V}_c,\\ u_p^*(v \circ T) &= \int_{\Omega_p} \hat{u}_p (v \circ T)\,\mathrm{d}x = \int_{\Omega_c} (\hat{u}_p \circ T^{-1}) v \,|\boldsymbol{J}_T^{-1}|\,\mathrm{d}x \quad \forall\, v \in \mathcal{V}_c.\end{aligned}$$

Defining $(u_p^* \circ T)(v) = u_p^*(v \circ T)$ as a functional on $\mathcal{V}_c$ allows us to set up a similar minimization problem to equation (2.6):

$$\begin{aligned}u_c^* &= \operatorname*{argmin}_{u_c^* \in \mathcal{V}_c^*} \frac{1}{2}\|u_c^* - u_p^* \circ T\|^2_{(L^2(\Omega_c))^*}\\ &= \operatorname*{argmin}_{\hat{u}_c \in \mathcal{V}_c} \frac{1}{2}\|\hat{u}_c - (\hat{u}_p \circ T^{-1})|\boldsymbol{J}_T|^{-1}\|^2_{L^2(\Omega_c)}.\end{aligned}$$

Again solving the associated Euler–Lagrange equation and expanding in terms of basis coefficients gives

$$\begin{aligned}\sum_{j=1}^{N_c} \hat{u}_c^j \int_{\Omega_c} \phi_c^j \phi_c^i \,\mathrm{d}x &= \sum_{j=1}^{N_p} \hat{u}_p^j \int_{\Omega_c} (\phi_p^j \circ T^{-1})\phi_c^i |\boldsymbol{J}_T|^{-1}\,\mathrm{d}x \quad \forall\, \phi_c^i \in \mathcal{V}_c\\ &= \sum_{j=1}^{N_p} \hat{u}_p^j \int_{\Omega_p} \phi_p^j (\phi_c^i \circ T)\,\mathrm{d}x \quad \forall\, \phi_c^i \in \mathcal{V}_c,\end{aligned}$$

which leads to a similar linear equality as above with the computational mass matrix $\boldsymbol{M}_c$,

$$\boldsymbol{M}_c\hat{\boldsymbol{u}}_c = \boldsymbol{C}^T\hat{\boldsymbol{u}}_p. \tag{2.8}$$

Interpreting the physical and computational mass matrices as Riesz maps on $\mathcal{V}_p$ and $\mathcal{V}_c$, respectively, we have

$$\boldsymbol{M}_p\hat{\boldsymbol{u}}_p = \boldsymbol{u}_p^* \quad \text{and} \quad \boldsymbol{M}_c\hat{\boldsymbol{u}}_c = \boldsymbol{u}_c^*,$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

allowing us to rewrite equation (2.8) as

$$\boldsymbol{u}_c^* = \boldsymbol{C}^T \boldsymbol{M}_p^{-1} \boldsymbol{u}_p^*,$$

implying that restriction is equal to the transpose of interpolation, $\boldsymbol{R} = \boldsymbol{P}^T$.

2.3.1. Coupled mass evaluation. Computation of the coupled mass matrix, $\boldsymbol{C}$, requires evaluating basis functions on both the physical and computational meshes and becomes slightly more involved when we do not have degrees of freedom that align between the two meshes. However, applying a change of variables to equation (2.7) via T, we observe that there are, indeed, two equivalent formulations of the weak form defining $\boldsymbol{C}$, i.e.,

$$[\boldsymbol{C}]_{ij} = \int_{\Omega_p} \phi_p^i (\phi_c^j \circ T)\, \mathrm{d}x = \int_{\Omega_c} (\phi_p^i \circ T^{-1}) \phi_c^j \, |\boldsymbol{J}_T^{-1}|\, \mathrm{d}x, \tag{2.9}$$

giving us some choice in how to evaluate the integral.

We elect to compute the integral in computational space, as our above assumption of using uniformly spaced grids on the computational domain greatly reduces the required computational effort, as it can be written in terms of evaluation of tensor-product elements on an axis-aligned grid. The algorithm proceeds as follows: We iterate through each element $\tau_p^i \in \mathcal{T}_p$ on the physical mesh and compute $\tilde{\tau}_p^i \subset \Omega_c$ by mapping the *vertices* of τ_p^i onto Ω_c through T. From this, we determine the subset $W_i = \{\tau_c^j : \tau_c^j \cap \tilde{\tau}_p^i \neq \emptyset\}$ of computational elements that overlap with $\tilde{\tau}_p^i$; in practice, this is done in a process that is similar to rasterization in computer graphics.

From this overlapping set, we loop through pairs of overlapping elements on both domains, (τ_p^i, τ_c^j), and compute a partial contribution of the integral in equation (2.9). The quadrature points on τ_c^j are mapped to a reference element on the physical mesh, $\hat{\tau}_p$. Quadrature points that lie outside of the reference element (which correspond to regions where the elements do not intersect when there is only partial overlap) are simply evaluated as 0, whereas points on the interior are used to evaluate the weak form as usual.

By determining overlapping sets of elements, we evaluate $\boldsymbol{C}$ in a way that is stable when one mesh is *highly refined* compared to the other. If we were to simply loop over elements on one mesh and map their quadrature points to the other domain, quadrature aliasing may occur, where certain elements do not receive any mass contribution even though the elements themselves geometrically intersect.

In our implementation, we evaluate and store $\boldsymbol{C}$ as a sparse matrix. However, we note that an implementation variant (and perhaps, more friendly to GPUs) is to precompute the overlapping sets W_i and compute the action of $\boldsymbol{C}$ on a vector via a matrix-free algorithm.

2.4. Coarse-grid scaling. Preliminary numerical experiments showed that, when using low-order numerical quadrature to evaluate $\boldsymbol{C}$, a significant "mis-scaling" could arise between the discretized problem on the computational grid and the original one on the physical grid. This leads to slow convergence, where the *direction* of the coarse-grid correction is a good one, but the scaling of the correction is poorly chosen relative to the current approximation, $\boldsymbol{u}_p$. This was observed to be particularly prevalent when the map T is ill-conditioned and has an oscillatory Jacobian.

To address this mis-scaling we use a greedy per-iteration rescaling of the coarse-grid correction. That is, we minimize the solution error of the coarse-grid correction in the $\boldsymbol{A}_p$-norm,

$$\min_{\gamma} \|\boldsymbol{u}^* - (\boldsymbol{u}_p + \gamma \boldsymbol{P} \boldsymbol{e}_c)\|_{\boldsymbol{A}_p}^2, \tag{2.10}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

where $\boldsymbol{u}^*$ is the exact solution to $\boldsymbol{A}_p\boldsymbol{u}^* = \boldsymbol{f}_p$. The value of γ that minimizes equation (2.10) is given by

$$\gamma^* = \frac{\langle \boldsymbol{u}^* - \boldsymbol{u}_p, \boldsymbol{P}\boldsymbol{e}_c\rangle_{\boldsymbol{A}_p}}{\|\boldsymbol{P}\boldsymbol{e}_c\|^2_{\boldsymbol{A}_p}} = \frac{\langle \boldsymbol{r}_p, \boldsymbol{P}\boldsymbol{e}_c\rangle_2}{\langle \boldsymbol{A}_p\boldsymbol{P}\boldsymbol{e}_c, \boldsymbol{P}\boldsymbol{e}_c\rangle_2}.$$

Computing γ^* requires two extra vector inner products and a single matrix–vector product with the physical stiffness operator, $\boldsymbol{A}_p$. The residual and interpolated error correction, $\boldsymbol{r}_p$ and $\boldsymbol{P}\boldsymbol{e}_c$, are reused from the computation of the coarse-grid correction itself, so that $\boldsymbol{A}_p\boldsymbol{P}\boldsymbol{e}_c$ is the only additional vector to be stored.

One consequence of this approach to computing γ is that it yields a different solver at each multigrid iteration. If the method is used as a preconditioner for a Krylov method (not considered here), then use of a flexible variant, such as FGMRES [31], would be needed.

3. Mappings. The solver outlined in Section 2.2 requires the mapping, T, from the physical domain to the computational domain as input. For many domains, such mappings to a square are known a priori. However, for more complicated geometries, such a mapping is likely not available. We first provide background theory on mappings between domains, motivating the learning of the mapping discussed in Section 3.1.

DEFINITION 3.1 (Harmonic map). *For some open, connected sets $U, V \subset \mathbb{R}^d$, a homeomorphism $T : \overline{U} \to \overline{V}$ is said to be* harmonic *if its components each individually satisfy Laplace's equation on the interior, i.e.,*

$$-\Delta T(\boldsymbol{x}) = \boldsymbol{0} \quad \forall\, \boldsymbol{x} \in U.$$

Additionally, if we have a boundary homeomorphism $\phi : \partial U \to \partial V$ and let T satisfy the boundary condition

$$T|_{\partial U} = \phi, \tag{3.1}$$

then we refer to T as the harmonic extension *of ϕ.*

Harmonic maps arise from minimization of the Dirichlet energy of the map, T,

$$E_d[T] = \int_\Omega \|\nabla T\|_2^2 \,\mathrm{d}x,$$

constrained by the boundary condition in equation (3.1). As they are critical points of the functional, they can be considered generalizations of geodesics to maps between manifolds, in that they encode the shortest paths between points, given a distance metric. In this work, we seek to use harmonic maps, as they tend to have smooth structure and well-behaved derivatives.

Specifically, it is possible to construct a harmonic map, T, that is diffeomorphic on $\Omega_p \to \Omega_c$ in two dimensions. While we are aware of no equivalent explicit guarantee that such a T exists in higher dimensions, the overall framework is still applicable in higher dimensions if a suitable map, T, can be found. Here, we consider [1, Theorem 4].

THEOREM 3.2. *Let $U, V \subset \mathbb{R}^2$ be bounded, simply connected, open convex sets, and $\phi : \partial U \to \partial V$ a homeomorphism continuously mapping between the boundaries of U and V. If T is a harmonic extension of ϕ satisfying*

$$\begin{aligned} -\Delta T &= \boldsymbol{0} \quad \text{in } U, \\ T &= \phi \quad \text{on } \partial U, \end{aligned}$$

then T is a homeomorphism of $\overline{U}$ onto $\overline{V}$. Moreover, if the Jacobian determinant is strictly positive, $|\boldsymbol{J}_T(\boldsymbol{x})| > 0$ for all $\boldsymbol{x} \in U$, then T is smooth (and thus diffeomorphic) on the interior.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

REMARK 3.3. From [1], the boundary map ϕ being unimodal in each coordinate is a sufficient condition to ensure the Jacobian determinant is strictly positive on the interior.

With careful construction of the boundary map ϕ to ensure unimodality (i.e., boundary orientation is preserved, ϕ is injective, etc.), the map $T^{-1} : \Omega_c \to \Omega_p$ for convex Ω_p can be guaranteed to be smooth in two dimensions. For example, a mapping from a square to a smoothed triangle can be seen in Figure 3.1.

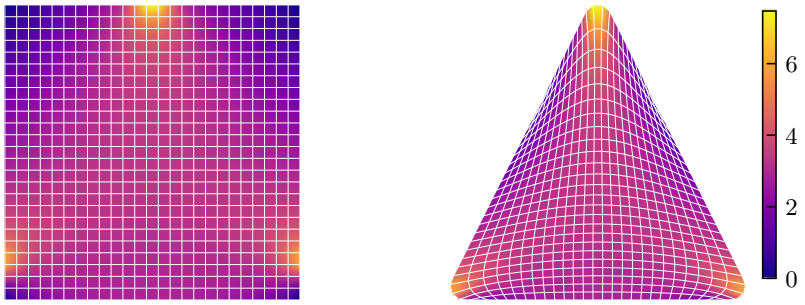


FIG. 3.1. *An example square domain (left) mapped to a smooth triangle domain (right) by solving the associated Laplacian. Shading on the domains denotes values of the Jacobian determinant, which only vanishes on the boundary, implying that the map is smooth on the interior.*

3.1. Learned mappings. While harmonic maps can offer smooth, diffeomorphic transformations under suitable conditions, in practice we consider domains either where convexity is not satisfied or where the analytic boundary map cannot be constructed. To this end, we extend our framework to more complex domains by instead learning an approximate harmonic mapping, $T_\theta : \Omega_p \to \Omega_c$, parameterized by some learnable values θ.

In machine learning, neural ordinary differential equations [8] (neural ODEs or NODEs) are a continuous generalization of feed-forward networks wherein the layer update is given by some continuous function $\boldsymbol{f}(\boldsymbol{x}, t) : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$; evaluation of the neural network is then computed by integrating $\boldsymbol{f}$ in time. In this context, we use neural ODEs as they guarantee a function that is smooth and invertible (under loose assumptions)—sufficient for use as our domain mapping.

Formally, we have the ODE given by

$$\frac{\mathrm{d}\boldsymbol{z}}{\mathrm{d}t}(t) = \boldsymbol{f}_\theta(\boldsymbol{z}(t;\boldsymbol{x}), t), \tag{3.2}$$

$$\boldsymbol{z}(0;\boldsymbol{x}) = \boldsymbol{x}, \tag{3.3}$$

where $\boldsymbol{f}_\theta(\boldsymbol{x}, t)$ is a learned vector field with parameters θ and $\boldsymbol{z}(t;\boldsymbol{x})$ is the latent state of the ODE—we use the notation $\boldsymbol{z}(\cdot;\boldsymbol{x})$ to make explicit the initial condition, $\boldsymbol{x}$, in the notation. Rewriting this in integral form, we obtain an expression for the mapping as

$$T_\theta(\boldsymbol{x}) = \boldsymbol{z}(1;\boldsymbol{x}) = \boldsymbol{x} + \int_0^1 \boldsymbol{f}_\theta(\boldsymbol{z}(t;\boldsymbol{x}), t)\,\mathrm{d}t, \tag{3.4}$$

where T_θ is now a learned map parameterized by θ. Derivatives of the network evaluation, for both back-propagation and finite-element assembly, can be efficiently computed by integration of an adjoint equation; see [8, 26] and Appendix A.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

The inverse of the map is computed by integrating backwards through the vector field. Denoting this backwards ODE solution by $\bar{\boldsymbol{z}}$, we can similarly define another ODE by

$$\begin{aligned} \frac{\mathrm{d}\bar{\boldsymbol{z}}}{\mathrm{d}t}(t) &= -\boldsymbol{f}_\theta(\bar{\boldsymbol{z}}(t;\boldsymbol{x}),t), \\ \bar{\boldsymbol{z}}(0;\boldsymbol{x}) &= \boldsymbol{x}, \end{aligned} \tag{3.5}$$

giving us the inverse mapping

$$T_\theta^{-1}(\boldsymbol{x}) = \bar{\boldsymbol{z}}(1;\boldsymbol{x}) = \boldsymbol{x} - \int_0^1 \boldsymbol{f}_\theta(\bar{\boldsymbol{z}}(t;\boldsymbol{x}),t)\,\mathrm{d}t.$$

In this work, we evaluate T_θ and T_θ^{-1} through a fixed RK4 time-stepping scheme with five time steps applied to the ODE in equations (3.2) and (3.5). We parameterize the vector field $\boldsymbol{f}_\theta$ by a feed-forward neural network with depth 3 and width 64; that is, we have three hidden layers each with a dimensionality of 64. We use the hyperbolic tangent function as our activation to retain smoothness.

To train T_θ to behave approximately like a harmonic mapping, we will define a composite PINNs-style [27] formulation, following Theorem 3.2. Restating this explicitly, we would like to satisfy

$$\begin{aligned} -\nabla\cdot(\omega(\boldsymbol{x})\nabla T_\theta(\boldsymbol{x})) &= \boldsymbol{0} \quad \text{in } \Omega_p, \\ T_\theta(\partial\Omega_p) &= \Omega_c, \end{aligned}$$

where $\omega : \mathbb{R}^d \to \mathbb{R}^+$ is a scalar weight function specifying the relative *importance* of a region in physical space: areas with higher values of $\omega(x)$ will subsequently be mapped to larger regions over the computational domain. For the scope of this work, however, we will fix $\omega \equiv 1$.

First, consider the boundary conditions of the map, which we impose weakly through a loss function inspired by large-deformation diffeomorphic metric mappings (LDDMMs) [20]. Let $I(\cdot;\Omega)$ be the unsigned distance function, defined as

$$I(\boldsymbol{x};\Omega) = \begin{cases} \|\boldsymbol{x} - P(\boldsymbol{x};\partial\Omega)\|_2 & \boldsymbol{x} \notin \Omega, \\ 0 & \text{otherwise}, \end{cases}$$

where $P(\boldsymbol{x};\partial\Omega) = \operatorname{argmin}_{\boldsymbol{y}\in\partial\Omega} \|\boldsymbol{y}-\boldsymbol{x}\|_2^2$ is the orthogonal projector onto the boundary of Ω. This is referred to as the *image* of the domain, and can be interpreted as encoding the interior volume and boundary of the underlying geometry, as shown in Figure 3.2. We emphasize that both of these functions are defined over all $\mathbb{R}^d$, not simply in their respective domains.

Next, we define the loss by

$$\ell(\theta) = \|I_p \circ T_\theta^{-1} - I_c\|_{L^2(\Omega_c)}^2 + \alpha\|\nabla\cdot(\omega\nabla T_\theta)\|_{L^2(\Omega_p)}^2, \tag{3.6}$$

where $I_p(\boldsymbol{x}) = I(\boldsymbol{x},\Omega_p)$ and $I_c(\boldsymbol{x}) = I(\boldsymbol{x},\Omega_c)$, so that the first term weakly enforces the boundary condition of the harmonic map. Intuitively, points should preserve their distance to the domain boundary after being transformed with T_θ. The second term in the loss encourages the resulting function to behave approximately like a harmonic: when $\omega(x) = 1$, we have that $\nabla\cdot(\omega\nabla T_\theta) = \Delta T_\theta \in \mathbb{R}^d$ is the componentwise Laplacian of T_θ and $\alpha > 0$ is a regularization parameter penalizing large Laplacian values. Computation of this componentwise Laplacian is detailed in Appendix A.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

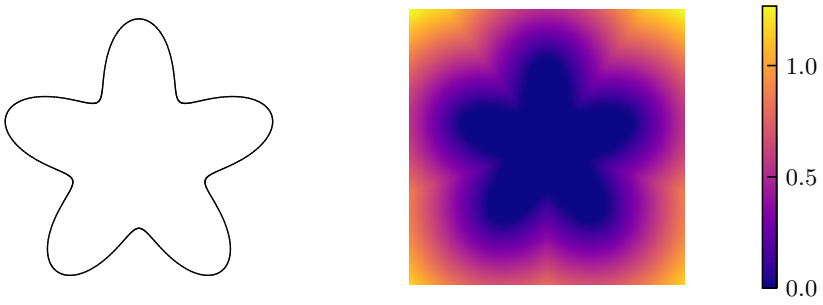


FIG. 3.2. *An example star-shaped domain (left) with corresponding unsigned distance function (right). Even if the underlying geometry is nonconvex it is possible to construct a distance function to represent it.*

The integral in equation (3.6) is approximated by a Monte Carlo scheme. For the *boundary matching* term, we sample points from three distinct geometric domains: a region extending slightly past the computational domain: $(-1.5, 1.5)^2$, the boundary of the computational domain $\partial\Omega_c$, and the mapped physical boundary $T_\theta^{-1}(\partial\Omega_p)$. For brevity, let $\mathcal{D} = \{(-1.5, 1.5)^2, \partial\Omega_c, T_\theta^{-1}(\partial\Omega_p)\}$ be the collection of these domains. For each domain, $d \in \mathcal{D}$, we draw N_B samples uniformly, denoted as $X_B^d = \{\boldsymbol{x}_i^d\}_{i=1}^{N_B}$. For the *regularization* term, we separately sample points over the physical domain, $X_R = \{\boldsymbol{x}_i^{\Omega_p}\}_{i=1}^{N_R} \sim \mathcal{U}(\Omega_p)$. This gives the discrete loss approximation,

$$\begin{aligned}\ell(\theta) &\approx \frac{1}{|\mathcal{D}|N_B} \sum_{d\in\mathcal{D}} \sum_{\boldsymbol{x}_c^d \in X_B^d} [I_p(T_\theta^{-1}(\boldsymbol{x}_c^d)) - I_c(\boldsymbol{x}_c^d)]^2 \\ &\quad + \frac{\alpha}{N_R} \sum_{\boldsymbol{x}_p \in X_R} \|\nabla \cdot (\omega(\boldsymbol{x}_p)\nabla T_\theta(\boldsymbol{x}_p))\|_2^2 \\ &= \frac{1}{|\mathcal{D}|N_B} \sum_{d\in\mathcal{D}} \sum_{\boldsymbol{x}_c^d \in X_B^d} L_B(\boldsymbol{x}_c^d;\theta) + \frac{\alpha}{N_R} \sum_{\boldsymbol{x}_p \in X_R} L_R(\boldsymbol{x}_p;\theta),\end{aligned}$$

where $L_B(\boldsymbol{x};\theta)$ and $L_R(\boldsymbol{x};\theta)$ are the partial local evaluations of the boundary and regularization terms, evaluated over the computational and physical domains, respectively.

Recall from above that our map, T_θ, is defined as the output of integrating through a learned vector field as in equation (3.4). Thus during training, to optimize our neural ODE, we compute

$$\begin{aligned}\frac{\partial \ell}{\partial \theta} &= \frac{1}{|\mathcal{D}|N_B} \sum_{d\in\mathcal{D}} \sum_{\boldsymbol{x}_c^d \in X_B^d} \frac{\partial T_\theta^{-1}(\boldsymbol{x}_c^d)}{\partial \theta} \frac{\partial L_B(\boldsymbol{x}_c^d;\theta)}{\partial T_\theta^{-1}(\boldsymbol{x}_c^d)} \\ &\quad + \frac{\alpha}{N_R} \sum_{\boldsymbol{x}_p \in X_R} \frac{\partial T_\theta(\boldsymbol{x}_p)}{\partial \theta} \frac{\partial L_R(\boldsymbol{x}_p;\theta)}{\partial T_\theta(\boldsymbol{x}_p)} \\ &= \frac{1}{|\mathcal{D}|N_B} \sum_{d\in\mathcal{D}} \sum_{\boldsymbol{x}_c^d \in X_B^d} \frac{\partial}{\partial \theta}\left(\boldsymbol{x}_c^d - \int_0^1 \boldsymbol{f}_\theta(\bar{\boldsymbol{z}}(t;\boldsymbol{x}_c^d), t)\,\mathrm{d}t\right) \frac{\partial L_B(\boldsymbol{x}_c^d;\theta)}{\partial T_\theta^{-1}(\boldsymbol{x}_c^d)} \\ &\quad + \frac{\alpha}{N_R} \sum_{\boldsymbol{x}_p \in X_R} \frac{\partial}{\partial \theta}\left(\boldsymbol{x}_p + \int_0^1 \boldsymbol{f}_\theta(\boldsymbol{z}(t;\boldsymbol{x}_p), t)\,\mathrm{d}t\right) \frac{\partial L_R(\boldsymbol{x}_p;\theta)}{\partial T_\theta(\boldsymbol{x}_p)}\end{aligned}$$

through back-propagation. By optimizing $\ell(\theta)$, we are also training the learned vector field f_θ, and thus the neural ODE, to output the map T_θ.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Learning the mapping for one domain took approximately 45 minutes, running 5000 training epochs on a machine with an NVidia H200 NVL GPU. While this is relatively expensive when compared to the cost of solving the resulting linear system itself, the mapping is computed only once for each domain (independently of the mesh). For example, if adaptive mesh refinement or a moving mesh method is used, then T_θ remains a valid mapping, as the domain itself does not change. This also allows for the training cost to be amortized for, e.g., time-stepped simulations or methods where a problem on the same domain is otherwise solved repeatedly.

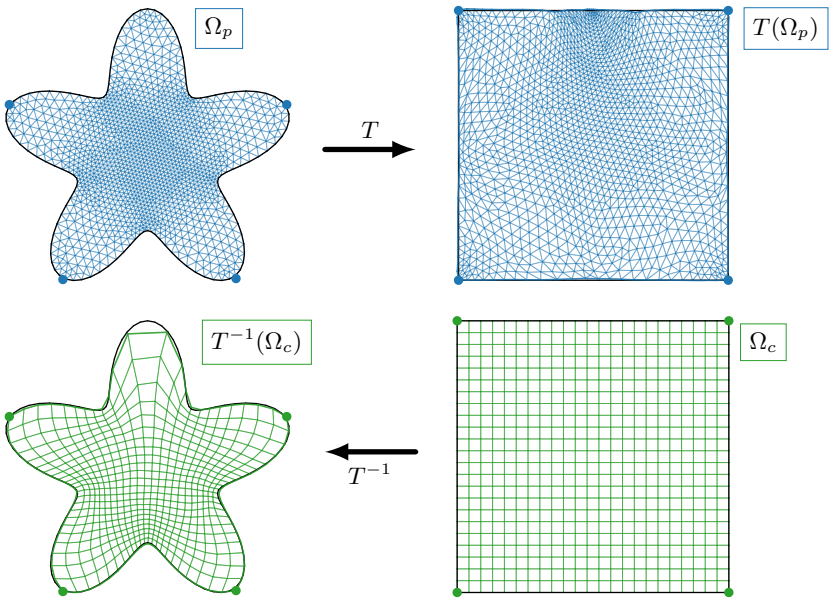


FIG. 3.3. *A learned mapping on the example star-shaped domain. On the left is the physical star domain, on the right is the square computational domain. The first row shows the vertices of the physical triangulation mapped to the computational domain through T, while the second row shows the vertices of the computational mesh mapped to the physical domain through T^{-1}. Due to the learning process, the mapping is only approximate; this can be seen on the domain boundaries.*

An example of a learned mapping on the star-shaped domain is shown in Figure 3.3. As the learned mapping is only approximate, there are several regions where, for example, the boundary is not mapped exactly.

4. Numerical results. In this section, we compare the convergence of our diffeomorphic multigrid (D-MG) solver to the "off-the-shelf" Ruge–Stüben algebraic multigrid (RS-AMG) solver [30], as implemented in PyAMG [3]. For all of these tests, we consider $\boldsymbol{A}_p$ to come from discretizing the physical-grid PDE with piecewise linear elements on triangles for a problem with a zero right-hand side for $\boldsymbol{f}_p$ and a random initial guess for $\boldsymbol{u}_p$. We use stationary iterations to solve until the Euclidean norm of the residual is reduced below the value 10^{-10}.

For the following results, we use $\nu = 2$ iterations of pre- and post-relaxation in both solvers. For the RS-AMG solver, we use forward and backward Gauss–Seidel, respectively, to maintain symmetry. For the D-MG solver, we use the same Gauss–Seidel relaxation on the finest grid and a red–black colored Gauss–Seidel on the coarse grids (the color ordering is swapped for pre- and post-relaxation to, again, maintain symmetry).

ETNA
Kent State University and
Johann Radon Institute (RICAM)

With our diffeomorphic multigrid solver, we now have the additional choice of selecting the size of the computational mesh as a solver parameter. This mesh size directly affects the resulting convergence and work done by the solver. We would like the computational cost of the V-cycle of our method to be as low as possible, while making the algorithm robust and efficient with the best possible convergence rate—these objectives are at odds, as a computational mesh that is too coarse will lead to poor corrections, while a much finer computational mesh will be too costly to run as a practical method. In this section, we fix the computational mesh so that the resulting solver does approximately the same amount of work as the RS-AMG solver, though we show the behavior of the method as this mesh size is varied in Section 4.2.

To make the relative work of our solver comparable to that of AMG, we aim to minimize the difference between the overall *grid complexity* (GC) of the solvers, defined as the sum of unknowns on all levels divided by the number of unknowns of the finest level,

$$\text{GC} = \frac{\sum_{\ell=1}^{L} N_\ell}{N_1},$$

where N_ℓ is the number of degrees of freedom (DoF) on level ℓ. We construct a comparable diffeomorphic solver by first running the RS-AMG setup on the physical problem to determine its grid complexity, then experimentally determining a uniform grid size that minimizes the difference of the resulting grid complexities, favoring higher complexity on the D-MG solver to lower. We also measure the operator complexity (OC) of the solvers as a metric of asymptotic storage cost,

$$\text{OC} = \frac{\sum_{\ell=1}^{L} \text{nnz}_\ell}{\text{nnz}_1},$$

where nnz_ℓ is the number of nonzeros in the stored operator on level ℓ. We note that we do not use this measure in the setup of our solver, but just to evaluate its efficiency.

4.1. Varying geometry and diffusivity. We test our solver against AMG on several domains in two dimensions, including both domains with known conformal analytic mappings and ones where the mapping must be learned. These domains and test problems will be detailed below, as well as their results summarized in Tables 4.1 and 4.2. Unless otherwise stated, we take $\mathcal{D} = I$. We demonstrate our diffeomorphic solver with and without coarse-grid scaling.

Rotated rectangle: a rectangle with dimensions $w = 5$ and $h = 2$ centered at the origin and rotated $\theta = \pi/6$ radians counterclockwise. This has an analytic linear mapping defined by

$$T(x, y) = \begin{bmatrix} 2/w & \\ & 2/h \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

This introduces a constant axis-aligned anisotropy in the computational problem. We note that the anisotropy induced is not so severe as to not be effectively handled by pointwise relaxation. Though, if the rectangle was further stretched, then line relaxation or semi-coarsening [11] would be necessary to retain convergence in geometric solvers.

Quarter annulus: a cross-section of an annulus extending from $\theta = \pi/2$ to π, with inner radius $r_2 = 1/2$ and outer radius $r_1 = 1$. This has a known mapping to the square domain given by

$$T(x, y) = \begin{bmatrix} 2(\sqrt{x^2 + y^2} - r_2)/(r_1 - r_2) - 1 \\ (4/\pi)\,\text{atan2}(y, x) - 3 \end{bmatrix}.$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Here, we use the two-argument arctangent function, $\mathrm{atan2}$, that returns $\arctan(y/x)$ when $x > 0$, but properly extends the range of arctangent from $-\pi$ to π for negative values of x.

Restricted channel: a domain consisting of the unit square, $(-1, 1)^2$, with two semicircle "cut-outs" of radius $r = 0.1$ located on the top and bottom edges, centered at $(0, 1)$ and $(0, -1)$, respectively. This can be seen as a stretched version of the classic restricted channel test problem in fluid dynamics with an equal aspect ratio. We learn the mapping, T, by a neural ODE.

Wavy box: a deformed, nonconvex box with boundary defined by

$$x(\phi) = [(|\cos\phi|^s + |\sin\phi|^s)^{-1/s}(W/2) + A\cos(k\phi)]\cos\phi + W/2,$$
$$y(\phi) = [(|\cos\phi|^s + |\sin\phi|^s)^{-1/s} + A\cos(k\phi)]\sin\phi,$$

for polar angle $-\pi < \phi \leq \pi$ and parameters $s = 500$, $W = 3$, $A = 0.08$, and $k = 6$. While it may be possible to construct a smooth map to the computational domain, we elect to learn it via a neural ODE.

Smooth star: a five-legged star with smoothed edges, with boundary defined by

$$x(\phi) = \tfrac{1}{2}[1 + \cos(5\phi/2)^2]\cos\phi,$$
$$y(\phi) = \tfrac{1}{2}[1 + \cos(5\phi/2)^2]\sin\phi,$$

for polar angle $-\pi < \phi \leq \pi$. This domain does not have an analytical conformal mapping, nor is it convex; using our methodology, we learn the mapping T parameterized by a neural ODE.

Discontinuous circular coefficients: a unit square domain with discontinuous diffusivity coefficient given by

$$\mathcal{D}(x, y) = \begin{cases} 100 & \sqrt{x^2 + y^2} < 3/5 \\ 1 & \text{otherwise} \end{cases}.$$

The physical domain is meshed to align with the circular discontinuity, while the computational domain is not. As the physical domain matches our computational domain here, we simply set T to be the identity.

Discontinuous checkerboard coefficients: a unit square domain with rapidly oscillating checkerboard diffusivity coefficients, given by

$$\mathcal{D}(x, y) = \begin{cases} 100 & (\lfloor 4x \rfloor + \lfloor 4y \rfloor) \equiv 0 \mod 2, \\ 1 & \text{otherwise,} \end{cases}$$

where we use the floor function to define where the coefficient is large. The physical domain is not meshed to align with the discontinuous coefficients. Again, because the domains coincide, the identity mapping is used.

Discontinuous thin-layer coefficients: a unit square domain with a single vertical "stripe" discontinuity, given by

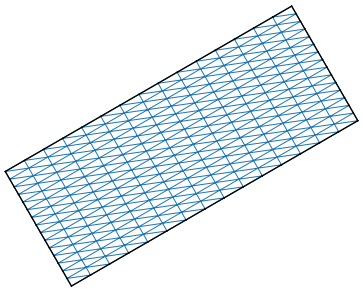
$$\mathcal{D}(x, y) = \begin{cases} 10^{-10} & 0 \leq x < \varepsilon, \\ 2 & \text{otherwise,} \end{cases}$$

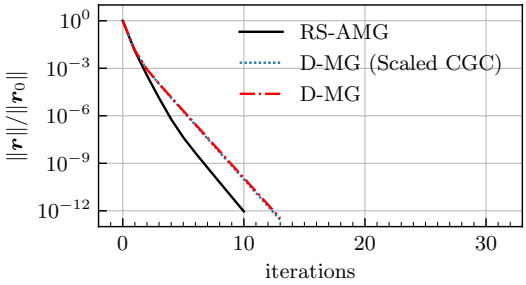
for $\varepsilon = 0.06$. The physical domain is not meshed to align with the discontinuous coefficients. Again, because the domains coincide, the identity mapping is used.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

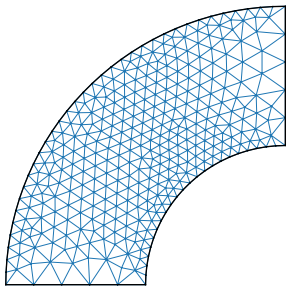
TABLE 4.1
Physical mesh (left) and residual plot (right). RS-AMG refers to a standard Ruge–Stüben AMG solver, while D-MG and D-MG (scaled CGC) refer to our diffeomorphic multigrid solver, without and with coarse-grid scaling, respectively.

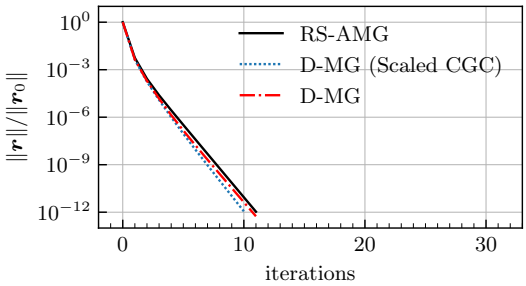
Rotated rectangle



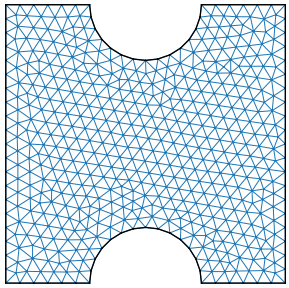


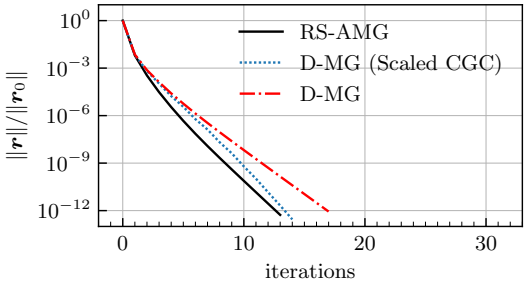
Quarter annulus



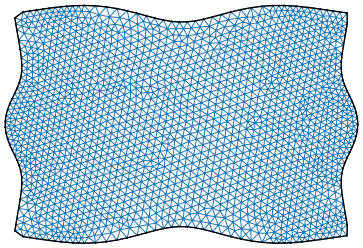


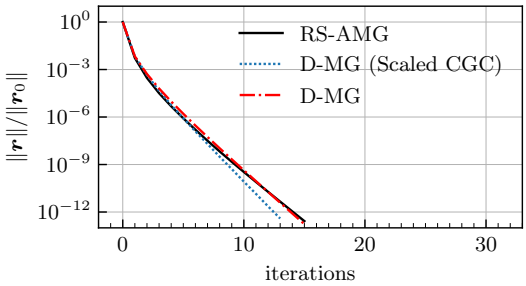
Restricted channel





Wavy box

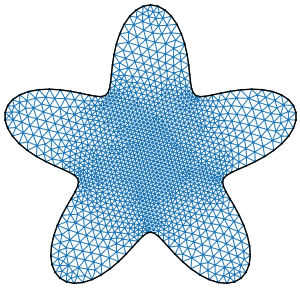


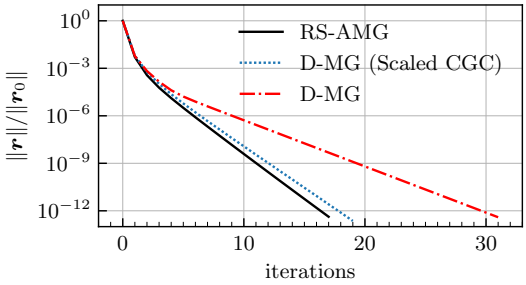


ETNA
Kent State University and
Johann Radon Institute (RICAM)

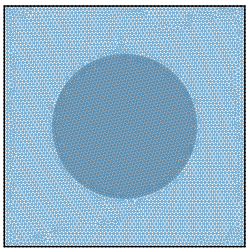
TABLE 4.2
Physical mesh (left) and residual plot (right). Rows 2–4 show the diffusivity in gray/white. RS-AMG refers to a standard Ruge–Stüben AMG solver, while D-MG and D-MG (scaled CGC) refer to our diffeomorphic multigrid solver, without and with coarse-grid scaling, respectively.

Smooth star

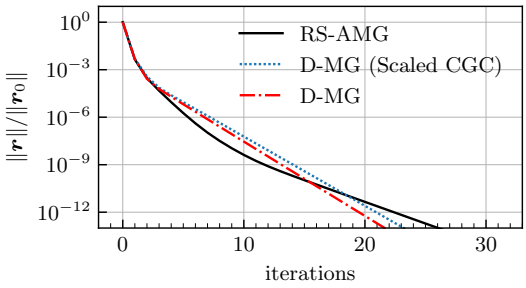




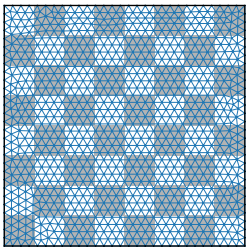
Discontinuous circular coefficients



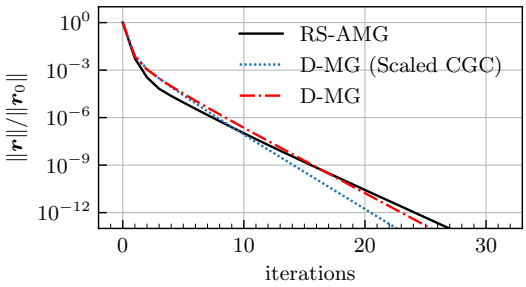
$\mathcal{D} = 1$ $\mathcal{D} = 100$



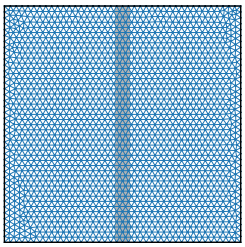
Discontinuous checkerboard coefficients



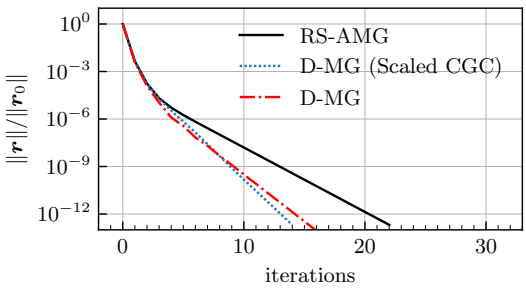
$\mathcal{D} = 1$ $\mathcal{D} = 100$



Discontinuous thin-layer coefficients



$\mathcal{D} = 2$ $\mathcal{D} = 10^{-10}$



ETNA
Kent State University and
Johann Radon Institute (RICAM)

In Tables 4.1 and 4.2, we display residual convergence results for standard Ruge–Stüben AMG (denoted RS-AMG) and compare to our diffeomorphic solver framework (denoted D-MG). We also include the residual for a diffeomorphic solver both with and without a scaled coarse-grid correction (see Section 2.4). For most of the domains for which analytic conformal mappings exist, the performance of the D-MG solver is comparable to that of the RS-AMG solver. For example, for the quarter annulus and discontinuous coefficient problems, we see equivalent or more rapid convergence. We note that the gain in convergence factor is not particularly significant except for the discontinuous circular coefficients and discontinuous thin-layer problems (five or so iterations). This is likely due to the geometric multigrid solver used, BoxMG, for the computational solve being robust when applied to problems with discontinuous coefficients. In contrast, for the rotated rectangle example, we observe slightly slower convergence (despite having an analytic mapping). This is due to the anisotropy that is induced onto the computational problem. Our geometric multigrid solver implementation uses only pointwise relaxation instead of, for example, alternating line relaxation, which would improve performance in this case [10].

The solver results are somewhat more varied for domains in which a learned mapping is applied. For the wavy box example, we achieve essentially identical behavior to RS-AMG, but we see slower convergence for D-MG applied to the smooth star and restricted channel domains. For the restricted channel domain, we lose about five iterations to convergence without scaling the coarse-grid correction but only one iteration with the scaling enabled. Without scaling the coarse-grid correction, D-MG requires about 15 more iterations for convergence on the smooth star domain, but this is improved to losing only two iterations with scaling. We propose two possible reasons for this degradation of convergence. First, these are both cases where we learn the mapping in an approximate sense and, for example, do not guarantee that we preserve key properties such as the area of the two domains, while the deformation for the wavy box case is much more simple (and, perhaps, easier to learn accurately). To address this, we could adapt the loss function for the neural ODE to emphasize appropriate geometric terms. Secondly, these are the two cases where the most deformation occurs in the mappings to the computational domain and, thus, the Jacobian of these maps is most ill-conditioned. It is possible that we could improve performance here by mapping to structured grids on domains other than the unit square, or by using domain-decomposition approaches to map submeshes to structured grids, but we consider these options beyond the scope of this original study.

To show how training the mapping affects the resulting multigrid solver, we selected several "snapshots" of the map during training of the learned mapping from the star-shaped domain to the computational domain. As shown in Figure 4.1, more training indeed improves the performance of the solver (without coarse-grid scaling) up to a point. Yet, running the training loop for too long can lead to a degradation in performance. This trend is also visible with the scaled coarse-grid correction. This is likely due to the fact that, while a decent map is necessary for a convergent multigrid solver, the optimization problem we are solving lacks any physical information from the solver itself; there is a potential mismatch in the objectives.

Additionally, we perform a study of the robustness of the learned mapping when the physical mesh is refined. We took the original mesh for the star-shaped domain (1509 DoF) and refined it twice to obtain meshes of sizes 2754 and 27 956, respectively. We evaluated the original learned mapping (trained on the mesh of size 1509) on all three of these refinements, then separately trained mappings (for 5000 epochs) for the larger 2754 and 27 956 sized meshes; these results are shown in Figure 4.2. While, in all cases, the unscaled preconditioner exhibits poor convergence (the behavior causing divergence over the domain is likely amplified at higher refinements), the preconditioner with a scaled coarse-grid correction obtains reasonable convergence at all refinements. It is interesting to note that retraining on the

ETNA
Kent State University and
Johann Radon Institute (RICAM)

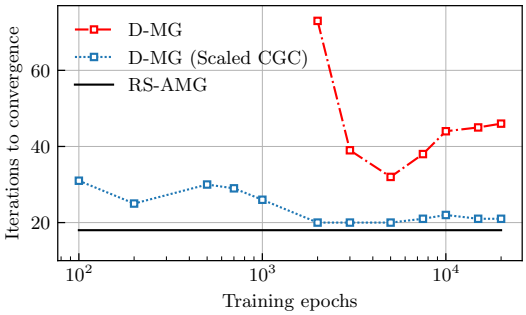


FIG. 4.1. *Effects of training duration versus resulting solver performance on the star-shaped domain. While more training epochs generally lead to a more convergent solver, an upward trend can be seen for the unscaled solver for a high number of training epochs. This is due to a potential mismatch between the objective function that was used to find the map, and what metrics actually constitute a "good" multigrid solver.*

higher-resolution meshes does not provide any meaningful boost to convergence in the scaled preconditioner; in fact, a slightly higher number of iterations for convergence is obtained. This could imply that the training for the higher-refinement meshes did not converge to as optimal a mapping as it did for the lowest-resolution mesh.

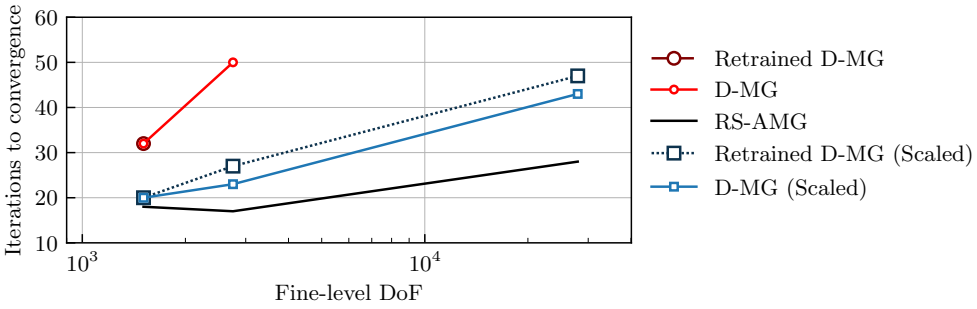


FIG. 4.2. *Convergence behavior of reusing the mapping from the original coarse star-shaped mesh on various mesh refinements of the star domain, compared with training a new map for the refined meshes. D-MG and D-MG (scaled CGC) refer to our solver using the original mapping trained at the lowest refinement. Retrained D-MG and Retrained D-MG (scaled CGC) refer to our solver using a mapping trained on that specific mesh refinement. Both the original and retrained D-MG variants without scaling fail to converge for higher refinements of the mesh; their data points are shown only for configurations that produce a convergent solver.*

We include grid and operator complexities for each solver in Table 4.3, along with grid sizes for the physical domains and those of the computational meshes. As noted above, we fix the size of the computational mesh for each problem to attain a similar grid complexity for our solver as is observed for the corresponding AMG solver. The resulting operator complexities of our diffeomorphic solvers, however, are always less than those of AMG. This is due to the bounded complexity in the Galerkin coarse-grid operators in BoxMG, having at most nine-point stencils in two dimensions. We remark that the operator complexities could be reduced further (in both cases) if a symmetric storage scheme was used, reducing the overhead to only five stored values per degree of freedom for BoxMG.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

TABLE 4.3
Physical degrees of freedom (N_p), number of degrees of freedom on the finest computational grid (N_c), coarse-grid scaling parameter (γ), grid complexity (GC), and operator complexity (OC) for the diffeomorphic MG (D-MG) and Ruge–Stüben AMG (RS-AMG) solvers.

		D-MG				RS-AMG	
	N_p	N_c	γ	GC	OC	GC	OC
Quarter annulus	281	144	0.93	1.49	1.56	1.41	1.62
Restricted channel	430	196	0.99	1.45	1.52	1.41	1.61
Wavy box	1708	576	1.00	1.38	1.46	1.41	1.69
Smooth star	1381	529	1.10	1.44	1.53	1.39	1.60
Disc. circular coefficients	5031	1764	2.03	1.43	1.53	1.44	1.81
Disc. checkerboard coefficients	938	361	1.36	1.44	1.52	1.42	1.56
Disc. thin-layer coefficients	1777	729	1.02	1.49	1.59	1.47	1.89

4.2. Varying size of computational grids. To study the effect of the resolution of the computational mesh, we selected the quarter annulus domain and varied the dimensions of the computational mesh, recording the resulting convergence factor and how it changes as a function of mesh size. The results of this are displayed in Figure 4.3, along with the relative work-per-digit accuracy (wpd), given by

$$\text{wpd}(N_x, N_y) \propto -\frac{N_x N_y}{\log_{10} \rho(N_x, N_y)},$$

where $\rho(N_x, N_y)$ is the convergence factor of the D-MG solver with an $N_x \times N_y$ computational mesh.

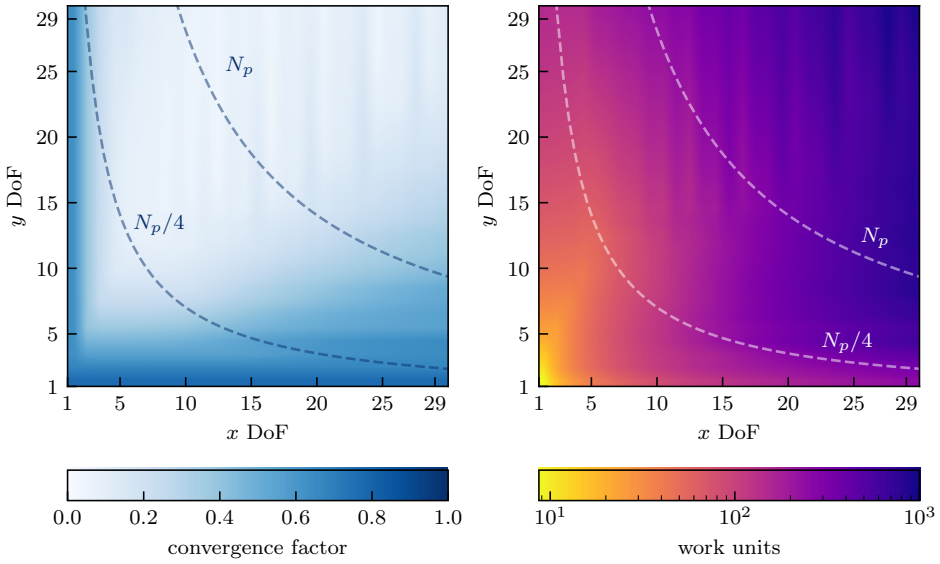


FIG. 4.3. *Convergence factor (left) and relative work-per-digit accuracy (right) of the D-MG solver on the quarter annulus domain as the resolution of the computational grid is varied. The x-axis indicates width of computational mesh (in elements), while y-axis indicates height. Dashed lines denote computational meshes with equivalent number of degrees of freedom as the physical mesh (N_p) and coarsening each spatial dimension by 2 ($N_p/4$).*

ETNA
Kent State University and
Johann Radon Institute (RICAM)

With this definition, we look for small values of wpd(N_x, N_y) to reflect efficient solvers. For this particular domain, while an equal aspect ratio usually suffices to give a convergent D-MG solver, having higher resolution in the y-axis will shift the resulting solver to have more rapid convergence; this behavior is due to our implementation of the map. The elongated portion of the quarter annulus is "straightened out" and mapped onto the y-axis of the computational domain. Hence, higher resolution in the y-axis allows the computational mesh to more accurately represent higher-frequency error modes. In general, one would expect higher resolution in both axes to result in better convergence factors, with the trade-off of having more work on the computational mesh. There exists a trade-off in convergence factor with work performed, as shown in the plot on the right of Figure 4.3: for the quarter annulus domain, best performance is achieved when approximately coarsening by 2 in each spatial direction, as evidenced by the work-per-digit accuracy plot.

4.3. Varying problem size. To demonstrate convergence scalability, we select the quarter annulus domain and vary the resolution of the physical mesh from 329 DoF to 1 066 204 DoF. The computational mesh sizes were selected by targeting a coarsening factor of $\alpha = 0.28$ with an aspect ratio of $N_y = 1.5N_x$, as observed in the results in Section 4.2. This is summarized by the equalities

$$N_x = \left\lceil \sqrt{\alpha N_p / 1.5} \right\rceil, \quad N_y = \lceil 1.5 N_x \rceil.$$

We solve this problem using our code, as described above and denoted D-MG below, and using the RS-AMG solver from HYPRE [13, 19] with default parameters.

From Figure 4.4, we see that the D-MG solver maintains consistent convergence factors and iteration counts as the problem size is refined. The same is not true for RS-AMG, where we see a strong increase in the number of iterations, from 27 for the smallest problem up to 46 for the largest, with corresponding convergence factors increasing from 0.36 to 0.51. While RS-AMG often achieves much better convergence factors for structured-grid discretizations of the Poisson problem on regular domains, this example shows how that behavior is somewhat anomalous, at least without substantial parameter tuning.

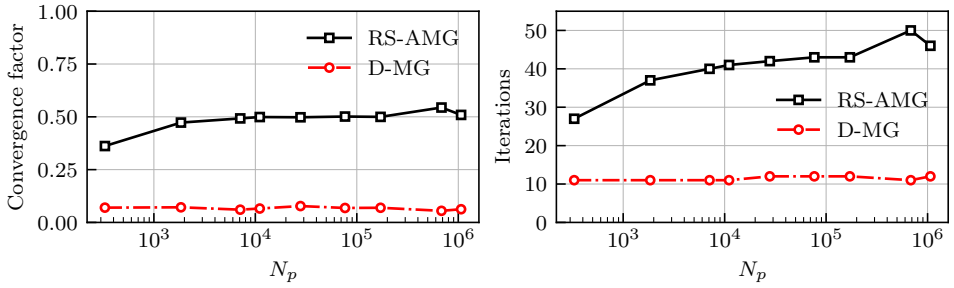


FIG. 4.4. *Convergence data for the D-MG and RS-AMG solvers as the problem size increases, indicating that both convergence factor and iterations to convergence remain stable for D-MG, but not RS-AMG, as the problem size increases.*

Figure 4.5 presents timing data for the scaling study and a breakdown of timings for the largest test case, both recorded running in serial on an Intel MacBook Pro with 2.4 GHz i9 processor and 32 GB of DDR4 RAM. Looking at the solver performance at left, the RS-AMG method tends to lead to slightly faster wall clock times for smaller problems, while our D-MG solver has faster runtimes for larger problems ($> 10^5$ DoF), with a speedup of

ETNA
Kent State University and
Johann Radon Institute (RICAM)

about 3 seconds (about 25%) on the largest problem. Breaking this down, this can be traced to the slightly longer setup time for the D-MG algorithm, which requires computing the geometric L^2 interpolation as described in Section 2.3. While our setup is slightly more involved, having to compute both the interpolation operator and rediscretize the physical PDE on the computational mesh, this is counteracted by a faster overall solve time, as seen in the right panel.

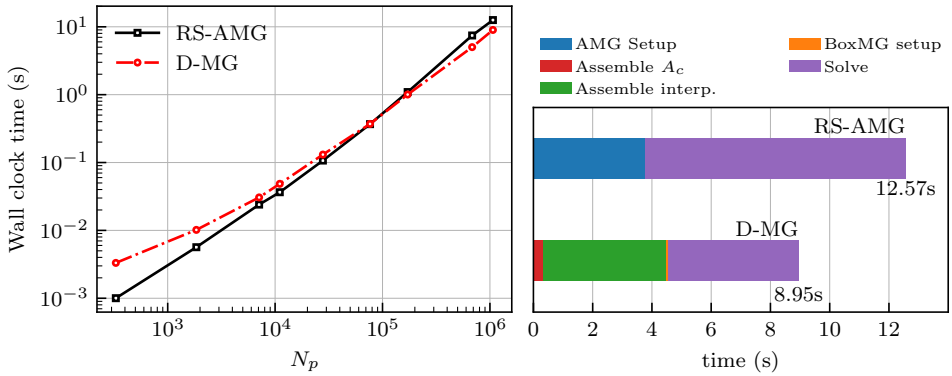


FIG. 4.5. *Timing data for the D-MG and RS-AMG solvers. At left, total time to solution with varying size of the physical mesh. At right, a breakdown of the timing for the largest problem size.*

It is clear that assembling the interpolation operator is the main computational bottleneck in the setup of our method, even though we do not invert the mass matrix directly. While our implementation reasonably uses sparse matrix operations to compute the interpolation operator, there are still optimizations that could be done to increase the efficiency of the operator assembly routine. For example, using a spatial tree data structure [4, 14] to more efficiently determine overlap of the physical and computational mesh elements, using a matrix-free approach to applying the operator, or even restructuring the routine to take advantage of parallelism are potential future directions to speed up the operator assembly. However, even with this slightly higher setup cost than traditional RS-AMG, the resulting speedup in solve times is enough to counteract this on larger problems. Notably, for the largest problems, we require about one-fourth of the number of iterations as RS-AMG to achieve a relatively strict solver tolerance, although our cost per iteration is higher than that of RS-AMG. This is due to the need to solve with the physical grid mass matrix twice per V-cycle (once for restriction from the physical grid to the finest computational grid, and once for interpolation back to the physical grid), done using just two iterations of unpreconditioned CG, which still incurs a similar cost as relaxation on the physical grid. Further approximating the mass solve, such as via mass lumping, may lead to improvements here.

5. Conclusions. We present a new framework for enabling the use of geometric multigrid on more complex geometries via the use of (learned) diffeomorphic mappings to transfer computational work to structured domains in an auxiliary space multigrid framework. This not only preserves the efficiency and scalability of robust geometric multigrid methods, but also allows their use on problems that are more traditionally solved by algebraic methods. Our method shows comparable performance to off-the-shelf AMG in terms of number of iterations to problem convergence. The diffeomorphic mapping approach allows us to reframe more difficult problem domains in a multilevel geometric setting, with the inherent benefits therein.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Possible future directions include validation of the use of learned mappings in this solver framework. It may very well be the case that there are certain *reference shapes* that are better suited to such computations: for example, mapping a circle domain to a square proves challenging due to the resulting singularities at the corners, but mapping to a different structured computational grid may be more feasible. Another interesting direction would be to consider applying this in a domain decomposition setting: geometric multigrid is already used in practice for semi-structured meshes, and allowing more flexibility in the types of (sub)domains would allow such solvers to be much more competitive with existing algebraic methods in terms of the geometries to which they can be applied. This would also more easily support adaptive refinement of the computational mesh where, for example, certain subregions require more resolution than others. Finally, exploring the implementation of this method on accelerators, such as GPUs, could result in significant speedup, both relative to the CPU implementation and algebraic methods on the GPU.

Acknowledgements. This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0003963. The work of S.M. was partially supported by an NSERC Discovery Grant.

This research used resources provided by the Darwin testbed at Los Alamos National Laboratory (LANL) which is funded by the Computational Systems and Software Environments subprogram of LANL's Advanced Simulation and Computing program (NNSA/DOE).

The authors gratefully thank the two anonymous reviewers for their work that strengthened this manuscript.

Appendix A. Adjoint computation of ODE Hessian. We assume we have some ODE solution, $\boldsymbol{z} : [0,1] \to \mathbb{R}^d$, that is at least C^2 with respect to both time and the initial condition ($\boldsymbol{x}$), and satisfies the ODE, i.e., equations (3.2) and (3.3). The Jacobian and Hessian of $\boldsymbol{z}$ with respect to the initial condition, $\boldsymbol{x}$, can be computed by integrating a system of adjoint equations.

We will first explicitly define the Jacobian matrix and Hessian tensor entrywise, as

$$[\boldsymbol{J}_z]_{ij} = \left[\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right]_{ij} = \frac{\partial z_i}{\partial x_j},$$
$$[\boldsymbol{\mathcal{H}}_z]_{ijk} = \left[\frac{\partial^2 \boldsymbol{z}}{\partial \boldsymbol{x}^2}\right]_{ijk} = \frac{\partial^2 z_i}{\partial x_j \partial x_k},$$

where $\boldsymbol{J}_z \in \mathbb{R}^{d\times d}$ and $\boldsymbol{\mathcal{H}}_z \in \mathbb{R}^{d\times d\times d}$ are assumed to be evaluated at some known time, t. Additionally, we will define the adjoint variables $\boldsymbol{W} = \boldsymbol{J}_z$ and $\boldsymbol{\mathcal{Q}} = \boldsymbol{\mathcal{H}}_z$. We will show that these can be computed by integrating an adjoint ODE system.

First, consider the evolution of the Jacobian, $\boldsymbol{W}$. Taking the derivative with respect to time, we obtain

$$\frac{\partial \boldsymbol{W}}{\partial t} = \frac{\partial}{\partial t}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}.$$

Because of our assumption on smoothness, we can interchange the order of the partial derivatives, giving

$$\begin{aligned}\frac{\partial \boldsymbol{W}}{\partial t} &= \frac{\partial}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{z}}{\partial t} = \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{f}_\theta(\boldsymbol{z}(t),t) \\ &= \frac{\partial \boldsymbol{f}_\theta}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{f}_\theta}{\partial \boldsymbol{z}}\boldsymbol{W}.\end{aligned} \tag{A.1}$$

The initial condition is given simply by $\boldsymbol{W}(0) = \boldsymbol{I}$, since $(\partial \boldsymbol{z}/\partial \boldsymbol{x})(0) = \partial \boldsymbol{x}/\partial \boldsymbol{x}$.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

A similar derivation can be done to find the time evolution of the Hessian, $\boldsymbol{\mathcal{Q}}$. Again, taking the derivative with respect to time, we obtain

$$\begin{aligned}\frac{\partial \boldsymbol{\mathcal{Q}}}{\partial t} &= \frac{\partial}{\partial t}\frac{\partial^2 \boldsymbol{z}}{\partial \boldsymbol{x}^2} = \frac{\partial^2}{\partial \boldsymbol{x}^2}\frac{\partial \boldsymbol{z}}{\partial t} \\ &= \frac{\partial^2}{\partial \boldsymbol{x}^2}\boldsymbol{f}_\theta(\boldsymbol{z}(t), t).\end{aligned}$$

Applying the second-order multivariate chain rule as given by Faà di Bruno's formula [15], we obtain the expression

$$\left[\frac{\partial \boldsymbol{\mathcal{Q}}}{\partial t}\right]_{ijk} = \sum_{l=1}^{d}\left(\frac{\partial f_i}{\partial z_l} q_{ljk}\right) + \sum_{l=1}^{d}\sum_{m=1}^{d}\left(\frac{\partial^2 f_i}{\partial z_l \partial z_m} w_{lj} w_{mk}\right). \tag{A.2}$$

The initial condition is given by the zero tensor, $\boldsymbol{\mathcal{Q}}(0)_{ijk} = 0$.

Computing $\boldsymbol{W}$ and $\boldsymbol{\mathcal{Q}}$ for some time, t, can then be done by integrating the coupled ODE system $(\boldsymbol{W}, \boldsymbol{\mathcal{Q}})$ using equations (A.1) and (A.2). The componentwise Laplacian of $\boldsymbol{z}$ can then be computed by taking traces of horizontal slices of $\boldsymbol{\mathcal{Q}}$,

$$[\Delta \boldsymbol{z}(t)]_i = \operatorname{tr}(\boldsymbol{\mathcal{Q}}_{i,:,:}(t)).$$

Note that element q_{ijk} of the stored Hessian depends only on elements q_{ljk} for $l = 1, \ldots, d$, meaning that to compute the Laplacian we, in fact, need to store and evolve the diagonal elements of only the last two indices (q_{ijj} for $i, j = 1, \ldots, d$).

REFERENCES

[1] G. Alessandrini and V. Nesi, *Univalent σ-harmonic mappings*, Arch. Ration. Mech. Anal., 158 (2001), pp. 155–171.
[2] N. Bell, S. Dalton, and L. N. Olson, *Exposing fine-grained parallelism in algebraic multigrid methods*, SIAM J. Sci. Comput., 34 (2012), pp. C123–C152.
[3] N. Bell, L. N. Olson, J. Schroder, and B. Southworth, *PyAMG: algebraic multigrid solvers in Python*, J. Open Source Softw., 8 (2023), Art. No. 5495, 5 pages.
[4] J. L. Bentley, *Multidimensional binary search trees used for associative searching*, Commun. ACM, 18 (1975), pp. 509–517.
[5] J. H. Bramble, J. E. Pasciak, and J. Xu, *The analysis of multigrid algorithms with nonnested spaces or noninherited quadratic forms*, Math. Comp., 56 (1991), pp. 1–34.
[6] E. Burman, S. Claus, P. Hansbo, M. G. Larson, and A. Massing, *CutFEM: discretizing geometry and partial differential equations*, Internat. J. Numer. Methods Engrg., 104 (2015), pp. 472–501.
[7] L. Chen, J. Wang, Y. Wang, and X. Ye, *An auxiliary space multigrid preconditioner for the weak Galerkin method*, Comput. Math. Appl., 70 (2015), pp. 330–344.
[8] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, *Neural ordinary differential equations*, in Proceedings of the 32nd International Conference on Neural Information Processing Systems, Curran Assoc., Red Hook, 2018, pp. 6572–6583.
[9] G. Chesshire and W. D. Hanshaw, *Composite overlapping meshes for the solution of partial differential equations*, J. Comput. Phys., 90 (1990), pp. 1–64.
[10] J. E. Dendy, *Black box multigrid*, J. Comput. Phys., 48 (1982), pp. 366–386.
[11] J. E. Dendy, S. F. McCormick, J. W. Ruge, T. F. Russell, and S. Schaffer, *Multigrid methods for three-dimensional petroleum reservoir simulation*, in SPE Symposium on Reservoir Simulation, Paper Number SPE-18409, Soc. Petro. Eng, Richardson, 1989.
[12] J. E. Dendy and J. D. Moulton, *Black box multigrid with coarsening by a factor of three*, Numer. Linear Algebra Appl., 17 (2010), pp. 577–598.
[13] R.D. Falgout et al., *HYPRE: high performance preconditioners*, Software Library. https://llnl.gov/casc/hypre, https://github.com/hypre-space/hypre.
[14] R. A. Finkel and J. L. Bentley, *Quad trees a data structure for retrieval on composite keys*, Acta Informatica, 4 (1974), pp. 1–9.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

[15] L. E. FRAENKEL, *Formulae for high derivatives of composite functions*, Math. Proc. Cambridge Philos. Soc., 83 (1978), pp. 159–165.
[16] I. M. GELFAND AND S. V. FOMIN, *Calculus of Variations*, Dover, Mineola, 2000.
[17] W. D. HENSHAW, *On multigrid for overlapping grids*, SIAM J. Sci. Comput., 26 (2005), pp. 1547–1572.
[18] W. D. HENSHAW AND D. W. SCHWENDEMAN, *An adaptive numerical scheme for high-speed reactive flow on overlapping grids*, J. Comput. Phys., 191 (2003), pp. 420–447.
[19] V. E. HENSON AND U. M. YANG, *BoomerAMG: a parallel algebraic multigrid solver and preconditioner*, Appl. Numer. Math., 41 (2002), pp. 155–177.
[20] M. HERNANDEZ AND U. R. JULVEZ, *Insights into traditional large deformation diffeomorphic metric mapping and unsupervised deep-learning for diffeomorphic registration and their evaluation*, Comput. Bio. Med., 178 (2024), Art. No. 108761, 31 pages.
[21] X. JIAO AND M. T. HEATH, *Common-refinement-based data transfer between non-matching meshes in multiphysics simulations*, Internat. J. Numer. Methods Engrg., 61 (2004), pp. 2402–2427.
[22] M. JUNTUNEN AND R. STENBERG, *Nitsche's method for general boundary conditions*, Math. Comp., 78 (2009), pp. 1353–1374.
[23] Z. LI, D. Z. HUANG, B. LIU, AND A. ANANDKUMAR, *Fourier neural operator with learned deformations for PDEs on general geometries*, J. Mach. Learn. Res., 24 (2023), Paper No. 388, 26 pages.
[24] Z. LI, N. KOVACHKI, K. AZIZZADENESHELI, B. LIU, K. BHATTACHARYA, A. STUART, AND A. ANANDKUMAR, *Fourier neural operator for parametric partial differential equations*, Preprint on arXiv, 2021. https://arxiv.org/abs/2010.08895
[25] A. PONT, R. CODINA, AND J. BAIGES, *Interpolation with restrictions between finite element meshes for flow problems in an ALE setting*, Internat. J. Numer. Methods Engrg., 110 (2017), pp. 1203–1226.
[26] L. S. PONTRYAGIN, *Mathematical Theory of Optimal Processes*, Gordon & Breach, Montreux, 1986.
[27] M. RAISSI, P. PERDIKARIS, AND G. E. KARNIADAKIS, *Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations*, Preprint on arXiv, 2017. https://arxiv.org/abs/1711.10561
[28] A. REISNER, M. BERNDT, J. D. MOULTON, AND L. N. OLSON, *Scalable line and plane relaxation in a parallel structured multigrid solver*, Parallel Comput., 100 (2020), Art. No. 102705, 13 pages.
[29] A. REISNER, L. N. OLSON, AND J. D. MOULTON, *Scaling structured multigrid to 500K+ cores through coarse-grid redistribution*, SIAM J. Sci. Comput., 40 (2018), pp. C581–C604.
[30] J. W. RUGE AND K. STÜBEN, *Algebraic multigrid*, in Multigrid Methods, S. F. McCormick, ed., vol. 3 of Frontiers Appl. Math., SIAM, Philadelphia, 1987, pp. 73–130.
[31] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2003.
[32] K. STÜBEN, *An introduction to algebraic multigrid*, in Multigrid, U. Trottenberg, C. Oosterlee, and A. Schüller, eds., Academic Press, San Diego, 2001, pp. 413–528.
[33] J. XU, *The auxiliary space method and optimal multigrid preconditioning techniques for unstructured grids*, Computing, 56 (1996), pp. 215–235.