

MOX-Report No. 69/2021

Accelerating Algebraic Multigrid Methods via Artificial Neural Networks

Antonietti, P.F.; Caldana, M.; Dede', L.

MOX, Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox-dmat@polimi.it

<http://mox.polimi.it>

Accelerating Algebraic Multigrid Methods via Artificial Neural Networks*

Paola F. Antonietti, Matteo Caldana, Luca Dede[†]

MOX, Dipartimento di Matematica, Politecnico di Milano,
Piazza Leonardo da Vinci 32, 20133 Milano, Italy

[†] Corresponding author: luca.dede@polimi.it

October 30, 2021

Abstract

We present a novel Deep Learning-based algorithm to accelerate – through the use of Artificial Neural Networks (ANNs) – the convergence of Algebraic Multigrid (AMG) methods for the iterative solution of the linear systems of equations stemming from Finite Element discretizations of Partial Differential Equations. We show that ANNs can be successfully used to predict the strong connection parameter that enters in the construction of the sequence of increasingly smaller matrix problems standing at the basis of the AMG algorithm, so as to maximize the corresponding convergence factor of the AMG scheme. To demonstrate the practical capabilities of the proposed algorithm, which we call AMG-ANN, we consider the iterative solution via the AMG method of the algebraic system of equations stemming from Finite Element discretizations of a two-dimensional elliptic equation with a highly heterogeneous diffusion coefficient. We train (off-line) our ANN with a rich data-set and present an in-depth analysis of the effects of tuning the strong threshold parameter on the convergence factor of the resulting AMG iterative scheme.

1 Introduction

In the last thirty years, there has been an increasing demand for computationally efficient methods to solve sparse linear system of equations stemming from numerical discretization of Partial Differential Equations (PDEs). For real-life

*P.F.A and L.D. are members of the INdAM Research group GNCS. P.F.A has been partially funded by the research project PRIN17, n.201744KLJL, by MIUR. P.F.A. and L.D. have been partially funded by the research project PRIN 2020, n.20204LN5N5, by MIUR.

problems, the typical size of the resulting algebraic systems makes direct or classical one-level methods impractical and hierarchical iterative solvers have been intensively developed and studied. This paper focuses on the Algebraic Multigrid (AMG) method ([47]) for the iterative solution of the symmetric and positive definite systems of equations stemming from Finite Element (FE) approximation ([23, 33, 34]) of elliptic partial differential equations. One of the main feature of AMG is that it is a purely matrix-based approach, thus it does not make use of any geometric information and the hierarchy of operators is constructed directly from the system matrix, provided that the underlying matrix has certain properties, see e.g., [7, 38, 42, 47]. AMG methods can be advantageous whenever geometric multigrid is not a viable option, e.g., whenever the sequence of coarser meshes at the basis of geometric multigrid is not available. AMG and AMG-like approaches have been developed to solve a variety of problems in the context of PDE-based simulations; here we mention, for example, the AMG method based on element interpolation (AMGe) for solving the discrete equations that arise in Ritz-type finite element methods, [9, 20], Maxwell’s equations [26], linear elasticity [4], Navier-Stokes’s equations [46] and multi-phase porous media [11]. In [5], AMG methods for large-scale supercomputing architectures are presented. In the paper [47] by Xu and Zikatanov, AMG methods are presented and analyzed in an unified framework and an abstract theory for the construction of optimal coarse space as well as quasi-optimal spaces is derived. The abstract framework of [47] covers most of the existing AMG methods, such as classical and energy-minimization AMG, unsmoothed and smoothed aggregation AMG, and spectral AMGe [13]. AMG methods for non-standard FE approximations have been also developed, for example in the context of Discontinuous Galerkin methods [2, 6, 40].

The AMG method relies on a set of parameters that defines how to algebraically carry out the coarsening phase. Often their tuning is based on experience and it could be rather inefficient in certain situations. In this paper, we propose using Machine and Deep Learning algorithms to make the choice of the AMG parameters fully automatic so as improve the efficiency of the method. The approach that we propose is based on the use of Artificial Neural Networks (ANNs). Artificial Neural Networks are Machine and Deep Learning models that are nowadays widely used in several problems in image recognition, speech recognition, and natural language processing [16]. The introduction of convolutional neural networks (CNNs) [29] changed modern object recognition process [17, 28, 39]. Today, the most advanced ANNs in image recognition are variations of CNNs: ResNet [18] and SENet [22].

Nowadays, Machine and Deep Learning models are increasingly being used in Scientific Computing [32], especially for the numerical approximation of ODEs and PDEs [31]. For example, Physics Informed Neural Networks (PINNs) have been introduced to approximate the solution of PDEs as a meshless method [35, 36] and ANNs are employed for model order reduction of parameter-dependent PDEs [15, 21, 37]. ANNs can also be employed to enhance the performances of algorithms and solvers used in “classical” numerical methods for the approximation of PDEs, i.e. as accelerators for Scientific Computing. In this context,

we mention for example: the enhancement of numerical stabilization methods for the FE approximation of advection-dominated differential problems, e.g. in [25, 44]; the use of ANN to optimally select artificial viscosity for Discontinuous Galerkin methods in [12]; exploiting CNN for grid refinement in Discontinuous Galerkin and Virtual Element methods in [1]; the hybrid ML-FETI-DP algorithm combining the advantages of adaptive coarse spaces in domain decomposition methods and certain supervised machine learning techniques that have been proposed in [19].

In this work, we make use of ANNs to improve the tuning of the strong threshold parameter that enters in the definition of AMG so as to improve its performance. In order to test the proposed approach, we consider a two-dimensional elliptic equation with a highly heterogeneous diffusion coefficient discretized by the FE method. In order to use the sparsity pattern of the underlying matrix as input of the neural network, we introduce a pooling operator. We show how an ANN-enhanced approach can effectively improve the AMG performance. The performance of the AMG method is measured in two ways: using the approximate convergence factor and using the elapsed time. We show that these two measures are strictly correlated, this entails that we have a unique way of measuring the performance. We demonstrate that, in some test cases, the value of the strong threshold parameter commonly used in literature can be improved so as to gain efficiency with respect to both measures. In particular, we test different models to tune the hyper-parameters of the model and we report the predictions of the models with the lowest loss.

The paper is structured as follows. In Section 2 we recall the basic elements of the AMG methods. In Section 3 we introduce the model problem and its FE discretization. In Section 4 we give a brief overview on ANNs. The results of the numerical experiments are showcased in Section 5, which is divided into two main parts. In the first one, we report a wide set of numerical experiments aimed at testing the performance when we tune the strong threshold parameter. In the second part, we design the architecture of a net, we introduce the pooling operator and test the model. Finally, in Section 6 we draw some conclusions.

2 Algebraic Multigrid Methods

In this section, we introduce the main ingredients of AMG methods; we refer the reader to [47] for a comprehensive description. We consider the linear system of equations:

$$\mathbf{A}_h \mathbf{u}_h = \mathbf{f}_h \tag{1}$$

where, for $n \in \mathbb{N}$, $\mathbf{A}_h \in \mathbb{R}^{n \times n}$ is symmetric positive definite, and $\mathbf{u}_h, \mathbf{f}_h \in \mathbb{R}^n$. Let $\mathcal{N}_h = \{1, \dots, n\}$ be the set with the indexes of all the variables. The set \mathcal{N}_h is split into two disjoint subsets \mathcal{C}_h and \mathcal{F}_h such that $\mathcal{N}_h = \mathcal{C}_h \cup \mathcal{F}_h$ and $\mathcal{C}_h \cap \mathcal{F}_h = \emptyset$.

Let $\mathbf{I}_H^h : \mathbb{R}^{n_H} \rightarrow \mathbb{R}^n$ be the interpolation operator that maps coarse level vectors into fine level vectors, and let $\mathbf{I}_h^H : \mathbb{R}^n \rightarrow \mathbb{R}^{n_H}$ be the restriction operator

that maps fine level vectors into coarse level vectors. It is assumed that \mathbf{I}_H^h can be written as:

$$(\mathbf{I}_H^h \mathbf{e}_H)_i = \begin{cases} (\mathbf{e}_H)_i & \text{if } i \in \mathcal{C}_h, \\ \sum_{k \in \mathcal{P}_i} w_{ij}^h (\mathbf{e}_H)_k & \text{if } i \in \mathcal{F}_h, \end{cases} \quad (2)$$

where $\mathbf{e}_H \in \mathbb{R}^{n_H}$ is a generic vector, $\mathcal{P}_i \subset \mathcal{C}_h$, for all $i \in \mathcal{F}_h$ is called a set of interpolatory variables for i and w_{ij}^h is a set of weights. Moreover, since \mathbf{A}_h is symmetric it is also assumed that:

$$\mathbf{I}_H^h = (\mathbf{I}_h^H)^\top. \quad (3)$$

Then, the coarse-level AMG matrix is defined as $\mathbf{A}_H = \mathbf{I}_h^H \mathbf{A}_h \mathbf{I}_H^h \in \mathbb{R}^{n_H \times n_H}$.

One of the key ingredients of the AMG method consists in the implementation of the interpolation operator \mathbf{I}_H^h previously described. The classical coarsening algorithm, which calls building \mathbf{I}_H^h , prescribes to maintain at the coarse level all the strong connections that are defined through a parameter θ , called the strong threshold parameter. Its rigorous definition is given in the following ([8]).

Definition 2.1 *Given a threshold $0 < \theta \leq 1$, the variable i strongly depends on the variable j if*

$$-(A_h)_{ij} \geq \theta \max_{k \neq i} (-(A_h)_{ik}).$$

As a matter of fact, performing the $\mathcal{C}_h/\mathcal{F}_h$ splitting and implementing the operators \mathbf{I}_h^H and \mathbf{I}_H^h requires choosing such strong threshold parameter θ . The performances of the AMG method will depend on this choice, which is empirically made a priori.

The last ingredient needed to define the AMG methods is a smoothing operator. In general one iteration of the smoothing can be written as:

$$\mathbf{u}_h^{(k+1)} = \mathbf{S}_h \mathbf{u}_h^{(k)} + (\mathbf{I}_h - \mathbf{S}_h) \mathbf{A}_h^{-1} \mathbf{f}_h = \mathbf{S}_h \mathbf{u}_h^{(k)} + \mathbf{g}_h, \quad k \geq 0, \quad (4)$$

where $\mathbf{S}_h \in \mathbb{R}^{n \times n}$ denotes the smoothing operator and \mathbf{I}_h is the identity operator. In the following, the notation:

$$\mathbf{u}_h^{(l)} = \text{smooth}^l(\mathbf{A}_h, \mathbf{u}_h^{(0)}, \mathbf{f}_h),$$

means that $\mathbf{u}_h^{(l)}$ is the result of l steps of (4), starting from an initial vector $\mathbf{u}_h^{(0)}$.

We report in Algorithm 1 one iteration of the two-level algorithm, where ν_1 and ν_2 are the smoothing steps that we apply before and after the error correction, respectively. The complete two-level AMG algorithm is outlined in Algorithm 2. As usual, in Algorithm 2, tol is a user-defined tolerance that is employed for a stopping criterion. Analogously, N_{max} is the maximum number of iterations allowed. We notice that the AMG algorithm involves selecting the parameter θ a priori.

Algorithm 1: One Iteration of the two-level AMG method

$\mathbf{u}_h^{(k+1)} = \text{two_level_iteration}(\mathbf{u}_h^{(k)}, A_h, \mathbf{f}_h, \nu_1, \nu_2, I_h^H, I_H^h)$

1 $\mathbf{u}_h^{(*)} \leftarrow \text{smooth}^{\nu_1}(A_h, \mathbf{u}_h^{(k)}, \mathbf{f}_h)$;
2 $\mathbf{r}_h \leftarrow \mathbf{f}_h - A_h \mathbf{u}_h^{(*)}$;
3 $\mathbf{r}_H \leftarrow I_h^H \mathbf{r}_h$;
4 $\mathbf{e}_H \leftarrow \text{solve}(A_H, \mathbf{r}_H)$;
5 $\mathbf{u}_h^{(*)} \leftarrow \mathbf{u}_h^{(*)} + I_H^h \mathbf{e}_H$;
6 $\mathbf{u}_h^{(k+1)} \leftarrow \text{smooth}^{\nu_2}(A_h, \mathbf{u}_h^{(*)}, \mathbf{f}_h)$

Algorithm 2: Two-Level AMG algorithm

$\mathbf{u}_h^{(k+1)} = \text{AMG}(\mathbf{u}_h^{(0)}, A_h, \mathbf{f}_h, \theta, \nu_1, \nu_2, N_{max}, tol)$

1 perform the $\mathcal{C}_h/\mathcal{F}_h$ -splitting using θ ;
2 build the operators I_h^H, I_H^h using θ ;
3 **while** $k < N_{max}$ **and** $\|A_h \mathbf{u}_h^{(k)} - \mathbf{f}_h\| / \|\mathbf{f}_h\| < tol$ **do**
4 $\mathbf{u}_h^{(k+1)} \leftarrow \text{two_level_iteration}(\mathbf{u}_h^{(k)}, A_h, \mathbf{f}_h, \nu_1, \nu_2, I_h^H, I_H^h)$
5 **end**

As a matter of fact, the two-level AMG Algorithm 2 can be immediately extended to many levels by simply calling recursively Algorithm 2 until a sufficiently coarse level is reached (where finally a direct linear solver is used). For the sake of the analysis carried out in the present paper, we will focus on the two-level method.

3 Model Problem

Throughout this work, we use standard notation for Sobolev spaces [30]. Let Ω be an open, bounded domain in \mathbb{R}^2 and let $\partial\Omega = \bar{\Gamma}_D \cup \bar{\Gamma}_N$, with $\overset{\circ}{\Gamma}_D \cap \overset{\circ}{\Gamma}_N = \emptyset$ and $\Gamma_D \neq \emptyset$. Then, the model problem reads:

$$\begin{cases} -\text{div}(\mu(x, y)\nabla u) = f & \text{in } \Omega, \\ u = g_D & \text{on } \Gamma_D, \\ \mu(x, y)\nabla u \cdot \hat{\mathbf{n}} = g_N & \text{on } \Gamma_N, \end{cases} \quad (5)$$

where $\hat{\mathbf{n}}$ is the outward directed unit normal vector, $f \in L^2(\Omega)$ is a given forcing term, and $g_D \in H^{1/2}(\Gamma_D)$, $g_N \in H^{-1/2}(\Gamma_N)$ are the given Dirichlet and Neumann boundary data, respectively. The function $\mu \in L^\infty(\Omega)$ is a positive diffusion coefficient. In this work it will be a piece-wise non-negative constant function.

To handle non homogeneous Dirichlet boundary condition we define, as usual, \tilde{u} by the means of the lifting $\tilde{u} = u - \tilde{g}$, where \tilde{g} is an extension of

g_D in $H^1(\Omega)$. The weak formulation of problem (5) reads:

$$\text{find } \tilde{u} \in H_{\Gamma_D}^1(\Omega) : (\tilde{u}, v) = F(v) \quad \forall v \in H_{\Gamma_D}^1(\Omega), \quad (6)$$

where $H_{\Gamma_D}^1(\Omega) := \{v \in H^1(\Omega) : v|_{\Gamma_D} = 0\}$ and

$$\begin{aligned} a(\tilde{u}, v) &:= \int_{\Omega} \mu(x, y) \nabla \tilde{u} \cdot \nabla v \, d\Omega, \\ F(v) &:= \int_{\Omega} f v \, d\Omega + \int_{\Gamma_N} h v \, d\gamma - \int_{\Omega} \mu(x, y) \nabla \tilde{g} \cdot \nabla v \, d\Omega. \end{aligned} \quad (7)$$

The well-posedness of problem (6) is guaranteed by the Lax-Milgram's theorem [10].

Now we pass to the FE formulation. We consider a quasi uniform mesh \mathcal{T}_h of Ω . We denote with the parameter $h > 0$ the mesh size of \mathcal{T}_h given by $h = \max_{T \in \mathcal{T}_h} h_T$, where h_T is the diameter of the element $T \in \mathcal{T}_h$. Then, the usual finite dimensional piecewise polynomial subspace of order $r \geq 1$ are defined as:

$$X_h^r = \{v_h \in C^0(\bar{\Omega}) : v_h|_T \in \mathbb{P}^r \quad \forall T \in \mathcal{T}_h\}, \quad r = 1, 2, \dots$$

In our case, we use:

$$V_h = \{v_h \in X_h^1 : v_h = 0 \text{ on } \Gamma_D\}.$$

The finite dimensional formulation of (6) reads:

$$\text{find } \tilde{u}_h \in V_h \text{ s.t.: } a(\tilde{u}_h, v_h) = F(v_h) \quad \forall v_h \in V_h. \quad (8)$$

By setting $n = \dim(V_h)$, we denote with $\{\phi_1, \dots, \phi_n\}$ the FE basis for V_h . Then, from Eq. (8), we obtain the linear system of equations $A_h \mathbf{u}_h = \mathbf{f}$, where:

$$a(\phi_j, \phi_i) = (A_h)_{ij}, \quad F(\phi_i) = (\mathbf{f})_i, \quad (\mathbf{u}_h)_i = \tilde{u}_i. \quad (9)$$

4 Artificial Neural Networks

An artificial neural network, ANN, is a regression (or classification) model which is simply a function $\mathcal{F} : \mathbb{R}^N \rightarrow \mathbb{R}^M$

$$\mathcal{F}(\mathbf{x}; \boldsymbol{\gamma}) = \mathbf{y}, \quad (10)$$

where \mathbf{x} is the input, \mathbf{y} is the predicted value of the regression and $\boldsymbol{\gamma}$ is the vector containing all the parameters of the model. The function \mathcal{F} is the composition of K functions $\mathcal{F}^{(k)}$ called *layers*, the number of layers K is called *depth* of the model. In the case of feed-forward neural network the layer is defined as

$$\begin{cases} \mathbf{a}^{(k)} = \mathbf{W}^{(k)} \mathbf{x}^{(k-1)} + \mathbf{b}^{(k)} \\ \mathbf{x}^{(k)} = \mathcal{H}^{(k)}(\mathbf{a}^{(k)}) \end{cases} \quad \text{for } k = 1, \dots, K \quad (11)$$

$\mathbf{x} = \mathbf{x}^{(0)}, \quad \mathbf{y} = \mathbf{x}^{(K)}, \quad N_0 = N, \quad N_K = M,$

where $W^{(k)} \in \mathbb{R}^{N_k \times N_{k-1}}$ (*weights*) and $\mathbf{b}^{(k)} \in \mathbb{R}^{N_k}$ (*biases*) are the parameters γ , and $\mathcal{H}^{(k)}(\cdot)$ is a scalar non-linear almost everywhere differentiable function that is applied component-wise to $\mathbf{a}^{(k)}$ and called *activation function*. The Rectified Linear Unit $\text{ReLU}(x) = \max\{0, x\}$ is our choice of activation function $\mathcal{H}^{(k)}(\cdot)$ since, in recent years, it has become very popular due to the fact that it greatly improves the convergence of the stochastic gradient descent algorithm compared to the sigmoid/tanh functions [28]. Moreover, it features lighter computations with a random initialization network as only about half of hidden units have a non-zero output and faster evaluation with respect to the sigmoid/tanh functions.

Next, We define the loss function \mathcal{L} . We assume that a dataset composed by P couples $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ is available; these are realizations of random variables \mathcal{X}, \mathcal{Y} . Once defined the ANN architecture, its training boils down to minimize the average training error, namely

$$J(\gamma) = \frac{1}{P} \sum_{i=1}^P \mathcal{L}(\mathcal{F}(\mathbf{x}^{(i)}; \gamma), \mathbf{y}^{(i)}). \quad (12)$$

A typical choice of the loss function \mathcal{L} that we also use in this paper is the mean square error (MSE).

For determining the parameters γ , we use the Adaptive Moment Estimation (Adam) method [27]. It is a variant of the stochastic gradient descent method that combines the Root Mean Squared propagation (RMSProp) algorithm [45] and momentum method [43] few other significant modifications, namely the momentum is recorded in the history of the gradient and there is a correction term of the bias for the estimation of the first and second order moments of the gradient.

Finally, to prevent overfitting and minimize the generalization error we employ four regularization techniques: early stopping (we stop the training at the point of smallest error with respect to the validation dataset), random parameter initialization, dropout [41, 17] and batch normalization [24].

As we want to use the matrix of the linear system A_h as input of the network we employ convolutional neural networks (CNN). Their characteristic is that the layer takes the form of a cross-convolution between the input and a matrix $K \in \mathbb{R}^{D \times D}$, called *kernel*. Three other hyper-parameters control how the convolution is performed: number of filters, stride and zero-padding size. Moreover, in the last stage of the layer a pooling function is applied. The pooling function is a form of down-sampling that replaces the output of the net at a certain location with a summary statistic of the nearby outputs. The aim of the pooling operation is to control the number of parameters and limit the overfitting. We refer the reader to [16] for more details.

5 ANN-enhanced AMG Method

This section is composed of two main parts. In the first one, we assess the relationship between the performance of the AMG methods and the strong

Algorithm 3: ANN-enhanced AMG

$$\mathbf{u}_h^{(k+1)} = \text{ANN_AMG}(\mathbf{u}_h^{(0)}, A_h, h, \mathbf{f}_h, \nu_1, \nu_2, N_{max}, tol, \gamma)$$

- 1 $V, C \leftarrow \text{pooling}(A_h, m)$ (Algorithm 4);
 - 2 $\bar{v} \leftarrow \sum_{i,j} (V)_{ij}$;
 - 3 $\sigma \leftarrow \sqrt{\frac{1}{m^2} \sum_{i,j} [(V)_{ij} - \bar{v}]^2}$;
 - 4 $(\hat{V})_{ij} \leftarrow \frac{(V)_{ij} - \bar{v}}{\sigma}$;
 - 5 $\theta^* \leftarrow \text{argmin}_{\theta} p \left(\mathcal{F}(\hat{V}, -\log_2(h), \theta; \gamma) \right)$;
 - 6 $\mathbf{u}_h^{(k+1)} \leftarrow \text{AMG}(\mathbf{u}_h^{(0)}, A_h, \mathbf{f}_h, \theta^*, \nu_1, \nu_2, N_{max}, tol)$ (Algorithm 2)
-

threshold parameter θ . In the second part, we design and use ANNs to predict the value of the strong threshold parameter θ^* that maximizes the performances of the AMG method.

We show in Algorithm 3 how we intend to use the prediction of the optimal strong threshold parameter θ^* realized by ANN within the AMG solver, which we call ANN-enhanced AMG algorithm. In particular, our approach determines θ^* to be used in the AMG algorithm starting from the matrix A_h and the mesh size h . This leverages on a map from a manipulation of A_h (\hat{V}), h and θ to a suitable performance index p of the AMG solver. Specifically, this map is realized by an ANN $\mathcal{F}(\mathbf{x}; \gamma)$ such that its inputs are $\mathbf{x} = (\hat{V}, -\log_2(h), \theta)$, while the output \mathbf{y} (the predicted value of the regression) coincides with a suitable performance index, say $\mathbf{y} = p(A_h, h, \theta)$, of the linear solver with AMG preconditioner. The steps in the ANN-enhanced AMG Algorithm 3 are the following:

- (1–4) as the matrix A_h can not be directly used as input of an ANN \mathcal{F} , suitable pooling and normalization steps are performed to assemble \hat{V} from A_h ($\hat{V} = \text{normalize}(\text{pooling}(A_h))$), where **pooling** is defined in (Algorithm 4) and **normalize** is defined at the end of Section 5.5.2);
- (5) the ANN \mathcal{F} built for the model problem is used to determine θ^* in order to minimize to the performance index p of the AMG;
- (6) the AMG Algorithm 2 is used with θ^* .

5.1 Numerical assessment of the strong threshold parameter θ

For our model problem (5) we select the diffusion coefficient μ to be a piece-wise non-negative constant function. We assume that μ features different patterns, where the domain Ω splits in strides or a checkerboard; see Figure 1. The value of $\mu(x, y)$ depends on which “tile” to which it belongs, namely

$$\mu(x, y) = \begin{cases} 1 & \text{if } (x, y) \in \Omega_{gray}, \\ 10^\epsilon & \text{if } (x, y) \in \Omega_{white}, \end{cases} \quad (13)$$

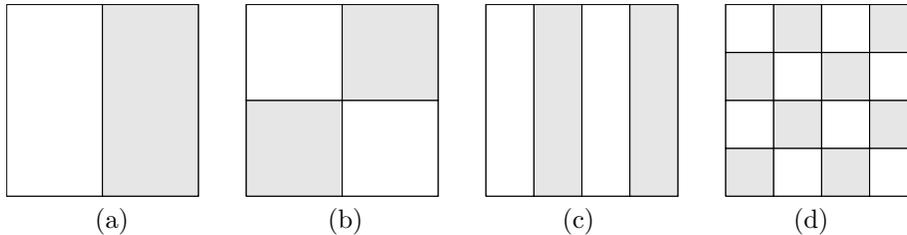


Figure 1: Four possible patterns of the diffusion coefficient μ of problem (5): it is defined such that $\mu = 1$ on the white tiles Ω_{white} and $\mu = 10^\varepsilon$ on the gray ones Ω_{gray} .

where ε is a parameter and Ω_{gray} and Ω_{white} are shown in Figure 1. The experiments were carried out so that the exact solution u of problem (5) is $u(x, y) = \cos(\pi x) \cos(\pi y)$ for patterns (a) and (b), while $u(x, y) = \cos(2\pi x) \cos(2\pi y)$ for patterns (c) and (d); Dirichlet boundary conditions are set on the whole boundary $\partial\Omega$. Moreover, we employ regular cartesian meshes, so that the discontinuity of μ is aligned with the mesh elements.

The implementation of the AMG method on which we rely on is the Boomer-AMG of the library HYPRE [14]. In particular, we use the AMG method as a preconditioner to accelerate the conjugate gradient (CG) iterative method [3].

To measure the performance of AMG we employ two performances indexes p : the elapsed CPU time and the approximate convergence factor ρ , defined as follows. Let $\rho^{(k)}$ be defined as

$$\rho^{(k)} = \left(\frac{\|\mathbf{r}^{(k)}\|}{\|\mathbf{r}^{(0)}\|} \right)^{\frac{1}{k}}, \quad (14)$$

where $\mathbf{r}^{(k)}$ is the residual at the k -th iteration and $\|\cdot\|$ is the standard euclidean norm. Then, we define ρ as

$$\rho = \rho^{(N_{it})},$$

where N_{it} is the number of iterations reached to reduce the (relative) residual below the given tolerance of the linear solver (here it is equal to $N_{it} = \min_k \{k \in \mathbb{N} \text{ such that } \|\mathbf{r}^{(k)}\| < 10^{-8}\}$).

5.2 Relation between θ and ρ

We investigate the relation between the strong threshold parameter θ and the corresponding approximated convergence factor ρ for a fixed test case (model problem), i.e. fixing the pattern of the diffusion coefficient μ , the coefficient ε and the size of the mesh h . Towards this goal, we have computed the value of ρ and the corresponding iteration counts as a function of the variables ε of the diffusion coefficient (Eq. (13)) and the mesh size h . The value of θ is kept fixed for each test.

By comparing one test with the others, we can determine if the different value of the strong threshold parameter θ has affected the convergence factor ρ of the linear solver. Six equally-spaced values of θ in $[0.12, 0.72]$ have been specifically chosen. We report the three most significant ones in Tables 1 and 2. The values of ε go from 0.0 (yielding the standard Laplacian problem with uniform diffusion), to 9.5, which produces a quite large discontinuity in the diffusion coefficient μ . The results reported in Table 1 have been obtained with a diffusion coefficient that has a “strides” pattern (Figure 1(c)), while Table 2 displays analogous results on the checkerboard pattern (Figure 1(d)).

From the results of Tables 1 and 2, it is clear that, if the choice of strong threshold θ is appropriate, there is almost always uniform convergence, independently from the mesh size h . This confirms that the AMG method works as intended also with a diffusion coefficient μ that presents large discontinuities, provided that θ is appropriately chosen.

The results reported in Tables 1 and 2 also show that for large values of the strong threshold parameter ($\theta = 0.72$), the approximate convergence factor ρ becomes larger, i.e. the method converges slower. We notice that this goes together with a larger number of iterations needed by the preconditioned CG to reduce the residual below the fixed tolerance. A possible explanation are kept at the following: consider that a larger θ means that more connections are kept at the coarse level, that is there are more \mathcal{C}_h -variables and thus A_H is larger. Moreover, recall that we set the AMG method to work on only two levels, and BoomerAMG does not solve the coarse system with a direct method, but instead uses an iterative method. Therefore it could be that the matrix A_H is associated with a grid still too fine to efficiently damp the low frequency modes, hence the convergence is slower.

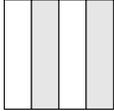
For the test cases that present less pronounced discontinuities, the value of $\theta = 0.24$ (which is almost the standard literature value) provides uniform convergence. On the other hand, we notice that in the strongly heterogeneous cases (i.e. when ε is large) deviating from the literature value of θ can result in a significant improvement in the approximate convergence factor.

5.3 Relation between θ and computational costs

In order to analyze more accurately the performance gain that can be obtained varying the strong threshold parameter θ , we computed the elapsed CPU time needed by the AMG preconditioned CG solver. Even with its limitations, this is still the quantity that in practice we want to minimize. Indeed, the aim of these test is to make sure that there could be some improvements in changing θ .

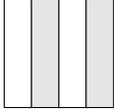
We report in Figures 2 and 3 the mean and standard deviation of the elapsed CPU time as a function of θ and for different values of ε (to determine μ). The results in Figures 2 and 3 have been obtained with the diffusion patterns reported in Figures 1(c) and (d), respectively. We point out that, as a single measurement is not reliable (especially when the elapsed time is small), each point displayed in Figures 2 and 3 is the mean of several evaluation of the solver time. In particular each simulation has been repeated 200, 100, 50, 20,

$\theta = 0.24$



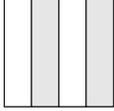
$\varepsilon \backslash h$	1.25e-01	6.25e-02	3.12e-02	1.56e-02	7.81e-03	3.91e-03	1.95e-03	9.77e-04
0.0	0.094(9)	0.071(8)	0.060(8)	0.054(8)	0.061(9)	0.063(9)	0.064(9)	0.066(10)
0.4	0.091(9)	0.069(8)	0.059(8)	0.057(8)	0.061(9)	0.062(9)	0.063(9)	0.066(10)
0.8	0.087(9)	0.066(8)	0.059(8)	0.058(8)	0.059(9)	0.060(9)	0.061(9)	0.068(10)
1.2	0.085(9)	0.066(8)	0.060(8)	0.059(8)	0.061(9)	0.060(9)	0.061(9)	0.069(10)
1.6	0.085(9)	0.065(8)	0.061(8)	0.067(9)	0.062(9)	0.061(9)	0.062(9)	0.070(10)
2.0	0.084(9)	0.065(8)	0.062(8)	0.068(9)	0.062(9)	0.061(9)	0.062(9)	0.070(10)
2.4	0.084(9)	0.065(8)	0.062(8)	0.068(9)	0.062(9)	0.062(9)	0.062(9)	0.070(10)
2.8	0.084(9)	0.065(8)	0.062(8)	0.069(9)	0.062(9)	0.062(9)	0.063(9)	0.070(10)
3.5	0.084(9)	0.065(8)	0.062(8)	0.069(9)	0.062(9)	0.062(9)	0.063(9)	0.070(10)
5.0	0.084(9)	0.065(8)	0.062(8)	0.069(9)	0.062(9)	0.062(9)	0.063(9)	0.070(10)
7.0	0.084(9)	0.065(8)	0.062(8)	0.069(9)	0.062(9)	0.062(9)	0.063(9)	0.070(10)
9.5	0.084(9)	0.065(8)	0.062(8)	0.069(9)	0.062(9)	0.062(9)	0.063(9)	0.070(10)

$\theta = 0.48$



$\varepsilon \backslash h$	1.25e-01	6.25e-02	3.12e-02	1.56e-02	7.81e-03	3.91e-03	1.95e-03	9.77e-04
0.0	0.094(9)	0.071(8)	0.060(8)	0.054(8)	0.061(9)	0.063(9)	0.064(9)	0.066(10)
0.4	0.091(9)	0.068(8)	0.057(8)	0.057(8)	0.059(9)	0.060(9)	0.061(9)	0.066(10)
0.8	0.087(9)	0.066(8)	0.059(8)	0.058(8)	0.059(9)	0.060(9)	0.061(9)	0.068(10)
1.2	0.077(8)	0.068(8)	0.059(8)	0.073(9)	0.063(9)	0.072(10)	0.089(11)	0.091(11)
1.6	0.076(8)	0.068(8)	0.059(8)	0.075(9)	0.063(9)	0.068(9)	0.083(11)	0.092(11)
2.0	0.076(8)	0.068(8)	0.059(8)	0.075(9)	0.063(9)	0.067(9)	0.085(10)	0.087(11)
2.4	0.075(8)	0.068(8)	0.059(8)	0.076(9)	0.063(9)	0.067(9)	0.084(10)	0.086(11)
2.8	0.075(8)	0.068(8)	0.059(8)	0.076(9)	0.063(9)	0.067(9)	0.084(10)	0.085(11)
3.5	0.075(8)	0.069(8)	0.059(8)	0.076(9)	0.062(9)	0.067(9)	0.083(10)	0.079(10)
5.0	0.075(8)	0.069(8)	0.059(8)	0.076(9)	0.062(9)	0.067(9)	0.083(10)	0.079(10)
7.0	0.075(8)	0.069(8)	0.059(8)	0.076(9)	0.062(9)	0.067(9)	0.083(10)	0.079(10)
9.5	0.075(8)	0.069(8)	0.059(8)	0.076(9)	0.062(9)	0.067(9)	0.083(10)	0.079(10)

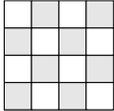
$\theta = 0.72$



$\varepsilon \backslash h$	1.25e-01	6.25e-02	3.12e-02	1.56e-02	7.81e-03	3.91e-03	1.95e-03	9.77e-04
0.0	0.094(9)	0.071(8)	0.060(8)	0.054(8)	0.061(9)	0.063(9)	0.064(9)	0.066(10)
0.4	0.054(7)	0.155(12)	0.112(10)	0.165(13)	0.150(13)	0.196(15)	0.189(15)	0.192(16)
0.8	0.047(7)	0.151(12)	0.107(10)	0.157(13)	0.153(13)	0.197(15)	0.172(14)	0.181(15)
1.2	0.052(7)	0.090(9)	0.108(10)	0.123(11)	0.136(12)	0.143(13)	0.169(14)	0.195(16)
1.6	0.054(7)	0.081(9)	0.110(10)	0.136(12)	0.132(12)	0.142(13)	0.160(14)	0.186(15)
2.0	0.054(7)	0.080(9)	0.109(10)	0.126(11)	0.133(12)	0.158(13)	0.168(14)	0.218(17)
2.4	0.054(7)	0.079(9)	0.110(10)	0.128(12)	0.137(12)	0.160(14)	0.170(14)	0.223(17)
2.8	0.054(7)	0.079(9)	0.111(10)	0.135(12)	0.139(12)	0.166(14)	0.171(14)	0.230(18)
3.5	0.054(7)	0.079(9)	0.112(10)	0.140(12)	0.143(12)	0.172(14)	0.171(14)	0.233(18)
5.0	0.054(7)	0.079(9)	0.112(10)	0.142(12)	0.144(12)	0.174(14)	0.172(14)	0.197(16)
7.0	0.054(7)	0.079(9)	0.112(10)	0.142(12)	0.144(12)	0.175(14)	0.172(14)	0.197(16)
9.5	0.054(7)	0.079(9)	0.112(10)	0.142(12)	0.144(12)	0.175(14)	0.172(14)	0.197(16)

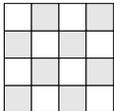
Table 1: Computed values of the approximate convergence factor $\rho = \rho^{(k)}$ (between brackets the AMG preconditioned CG iteration counts) w.r.t. parameters ε on rows and mesh size h on columns. In each table the the pattern of diffusion coefficient μ and the strong threshold θ (shown on the left) is fixed. The background color depends on ρ with colormap . It is scaled to range between minimum and maximum value (among all the tables) of ρ . Strides pattern for μ as in Figure 1(c).

$\theta = 0.24$



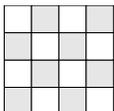
$\varepsilon \backslash h$	1.25e-01	6.25e-02	3.12e-02	1.56e-02	7.81e-03	3.91e-03	1.95e-03	9.77e-04
0.0	0.094(9)	0.071(8)	0.060(8)	0.054(8)	0.061(9)	0.063(9)	0.064(9)	0.066(10)
0.4	0.088(9)	0.070(8)	0.064(8)	0.067(9)	0.065(9)	0.064(9)	0.074(10)	0.075(10)
0.8	0.097(9)	0.103(10)	0.088(9)	0.101(10)	0.113(11)	0.125(12)	0.127(12)	0.134(13)
1.2	0.142(11)	0.160(12)	0.169(13)	0.171(13)	0.182(14)	0.194(15)	0.193(15)	0.205(16)
1.6	0.166(12)	0.196(13)	0.220(15)	0.228(16)	0.240(17)	0.255(18)	0.268(19)	0.277(20)
2.0	0.176(12)	0.221(14)	0.247(16)	0.261(17)	0.288(19)	0.302(20)	0.326(22)	0.344(24)
2.4	0.180(12)	0.234(15)	0.254(16)	0.289(19)	0.307(20)	0.326(22)	0.343(24)	0.362(25)
2.8	0.182(12)	0.236(15)	0.273(17)	0.294(19)	0.312(20)	0.333(22)	0.350(24)	0.375(26)
3.5	0.196(13)	0.237(15)	0.275(17)	0.297(19)	0.317(21)	0.333(23)	0.355(24)	0.386(27)
5.0	0.196(13)	0.238(15)	0.275(17)	0.298(19)	0.318(21)	0.333(23)	0.356(24)	0.388(27)
7.0	0.196(13)	0.238(15)	0.275(17)	0.298(19)	0.318(21)	0.333(23)	0.356(24)	0.388(27)
9.5	0.196(13)	0.238(15)	0.275(17)	0.298(19)	0.318(21)	0.333(23)	0.356(24)	0.388(27)

$\theta = 0.48$



$\varepsilon \backslash h$	1.25e-01	6.25e-02	3.12e-02	1.56e-02	7.81e-03	3.91e-03	1.95e-03	9.77e-04
0.0	0.094(9)	0.071(8)	0.060(8)	0.054(8)	0.061(9)	0.063(9)	0.064(9)	0.066(10)
0.4	0.088(9)	0.077(9)	0.064(8)	0.066(9)	0.065(9)	0.063(9)	0.063(9)	0.071(10)
0.8	0.097(9)	0.103(10)	0.088(9)	0.101(10)	0.113(11)	0.125(12)	0.127(12)	0.134(13)
1.2	0.113(10)	0.161(12)	0.107(10)	0.117(11)	0.137(12)	0.158(13)	0.184(15)	0.180(15)
1.6	0.129(10)	0.192(13)	0.118(11)	0.132(12)	0.140(12)	0.166(14)	0.191(15)	0.205(16)
2.0	0.147(11)	0.210(14)	0.124(11)	0.126(12)	0.152(13)	0.169(14)	0.194(15)	0.225(18)
2.4	0.150(11)	0.216(14)	0.127(11)	0.141(12)	0.155(13)	0.170(14)	0.184(15)	0.252(19)
2.8	0.152(11)	0.218(14)	0.125(11)	0.140(12)	0.144(12)	0.171(14)	0.196(16)	0.252(19)
3.5	0.153(11)	0.220(14)	0.126(11)	0.138(12)	0.152(13)	0.196(15)	0.203(16)	0.219(17)
5.0	0.153(11)	0.220(14)	0.126(11)	0.138(12)	0.153(13)	0.196(15)	0.211(16)	0.225(17)
7.0	0.153(11)	0.220(14)	0.126(11)	0.138(12)	0.153(13)	0.196(15)	0.211(16)	0.225(17)
9.5	0.153(11)	0.220(14)	0.126(11)	0.138(12)	0.153(13)	0.196(15)	0.211(16)	0.225(17)

$\theta = 0.72$



$\varepsilon \backslash h$	1.25e-01	6.25e-02	3.12e-02	1.56e-02	7.81e-03	3.91e-03	1.95e-03	9.77e-04
0.0	0.094(9)	0.071(8)	0.060(8)	0.054(8)	0.061(9)	0.063(9)	0.064(9)	0.066(10)
0.4	0.039(7)	0.141(11)	0.143(12)	0.172(13)	0.183(14)	0.195(15)	0.208(16)	0.215(17)
0.8	0.045(7)	0.119(10)	0.131(11)	0.173(13)	0.190(14)	0.213(16)	0.212(16)	0.240(18)
1.2	0.052(7)	0.115(10)	0.175(13)	0.159(13)	0.202(15)	0.240(17)	0.243(18)	0.266(20)
1.6	0.057(8)	0.108(10)	0.152(12)	0.171(13)	0.212(15)	0.244(17)	0.270(19)	0.274(20)
2.0	0.052(7)	0.115(10)	0.158(12)	0.169(13)	0.206(15)	0.222(16)	0.264(19)	0.266(19)
2.4	0.052(7)	0.115(10)	0.147(12)	0.153(12)	0.186(14)	0.221(16)	0.244(18)	0.273(20)
2.8	0.052(7)	0.116(10)	0.147(12)	0.162(13)	0.187(14)	0.224(16)	0.242(18)	0.264(19)
3.5	0.053(7)	0.117(10)	0.141(12)	0.162(13)	0.188(14)	0.235(17)	0.236(17)	0.260(19)
5.0	0.053(7)	0.117(10)	0.141(12)	0.162(13)	0.189(14)	0.244(17)	0.236(17)	0.259(19)
7.0	0.053(7)	0.120(10)	0.141(12)	0.162(13)	0.189(14)	0.245(17)	0.236(17)	0.259(19)
9.5	0.053(7)	0.120(10)	0.141(12)	0.162(13)	0.189(14)	0.245(17)	0.236(17)	0.259(19)

Table 2: Computed values of the approximate convergence factor $\rho = \rho^{(k)}$ (between brackets the AMG preconditioned CG iteration counts) w.r.t. parameters ε on rows and mesh size h on columns. In each table the the pattern of diffusion coefficient μ and the strong threshold θ (shown on the left) is fixed. The background color depends on ρ with colormap . It is scaled to range between minimum and maximum value (among all the tables) of ρ . Checkerboard pattern for μ as in Figure 1(d).

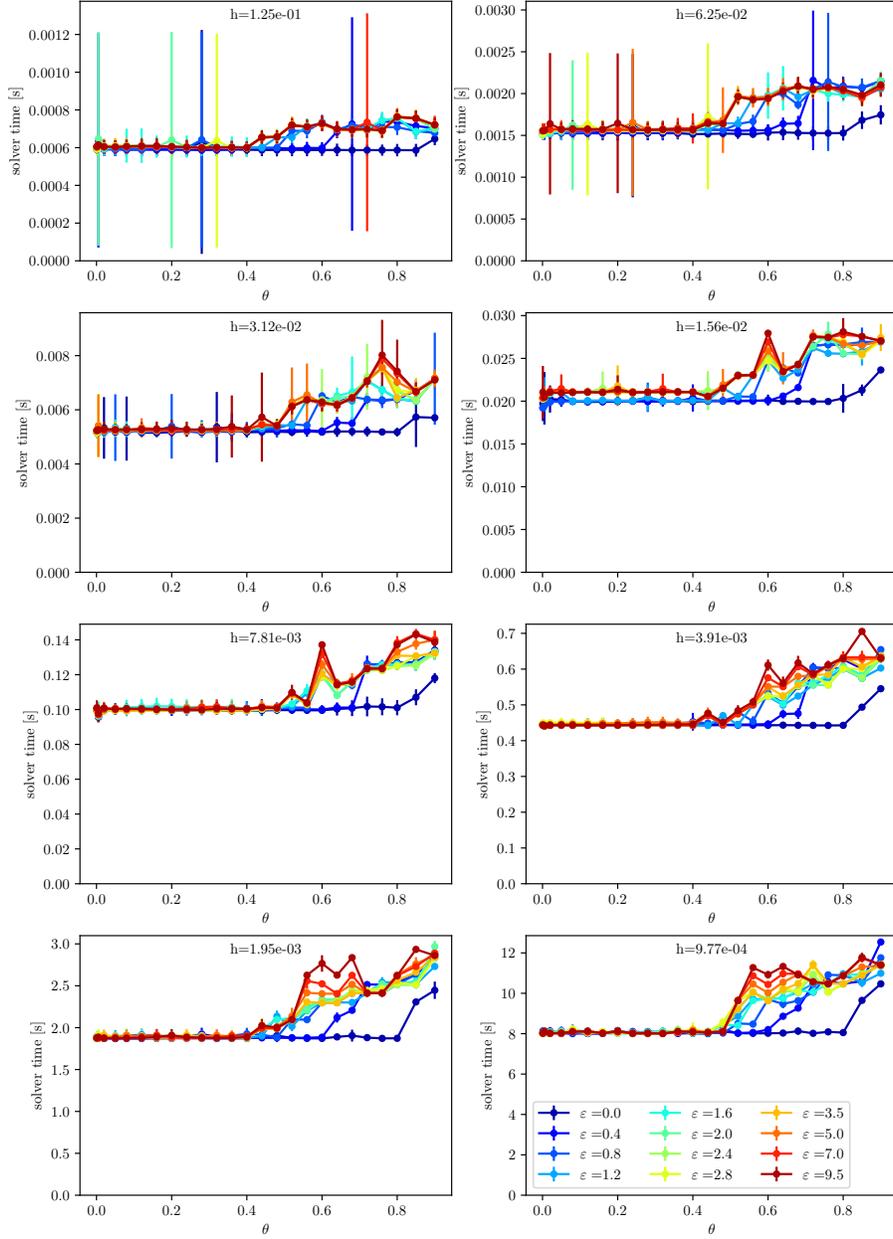


Figure 2: Mean and standard deviation of the elapsed CPU time to solve the linear system of equations (in seconds) based on employing the AMG preconditioned CG. In each plot we have fixed a different mesh size h . Each line represents a different parameter ε of the diffusion coefficient μ . The pattern of μ is reported in Figure 1(c).

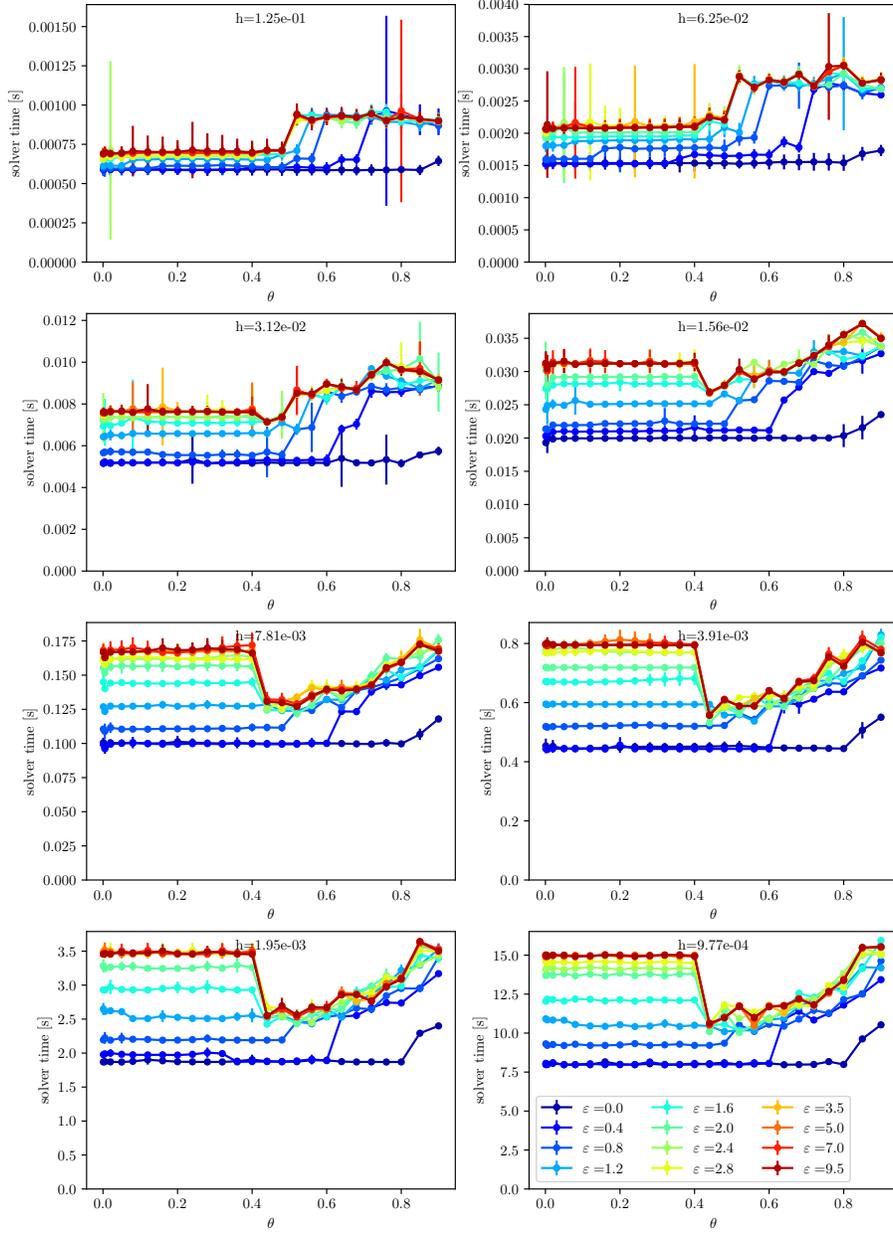


Figure 3: Mean and standard deviation of the elapsed CPU time to solve the linear system of equations (in seconds) based on employing the AMG preconditioned CG. In each plot we have fixed a different mesh size h . Each line represents a different parameter ε of the diffusion coefficient μ . The pattern of μ is reported in Figure 1(d).

10, 7, 5, and 4 times for each mesh refinement from the coarsest to the finest, respectively. The standard deviation has also been plotted.

The results shown in Figure 2 seem to indicate that choosing θ differently from the standard value suggested in literature ($\theta = 0.24$) does not result in any significant improvement. On the other hand, the plots of the four finest mesh refinements of Figure 3 reveal that a significant boost in the performance could be obtained. An optimal choice of the strong threshold could bring a speed up of up to 33% in terms of computational time.

We also observe that for small values of the strong threshold parameter ($\theta \leq 0.3$) there is an interval where the CPU time is almost constant: this appears to be true for all the test cases addressed in Figures 2 and 3. Since a smaller strong threshold parameter means that more connections are discarded in the coarsening phase, one would expect that as θ gets smaller, then the approximate convergence factor ρ may deteriorate, which in turn would lead to larger CPU times. A possible motivation of this behaviour is to consider that among the settings of BoomerAMG there is parameter that prevents the coarsening from being too small (in the present test its value has been set as default, i.e. equal to 1). Thus, the coarse system associated to A_H is still effectively damping the smooth components.

In conclusions, we have shown that, in some scenarios, there could be a significant performance improvement if we use optimized values of the strong threshold θ parameter. In the next section, we proceed to employ an ANN to make an a priori estimate of those values. The dataset that we built contains numerical simulations made with every combination of parameters among 8 mesh sizes h , 25 values of θ , 12 values of ε and 4 patterns of μ (Figure 1) for a total of 9600 samples.

5.4 Predicting the optimal strong threshold parameter θ by means of ANNs

Our goal is to design a model, namely an ANN, that enables predictions of the optimal strong threshold θ for a given model problem. We remark that, in the framework discussed in Section 3, fixing a test case (model problem) is equivalent to fixing the matrix A_h defined in Eq. (9). We define the optimal value of strong threshold parameter θ^* for a certain test case (problem) as the minimizers of a scalar performance index $p(A_h, h, \theta)$. Two possible choices for such performance index, which we previously discussed, are the approximate convergence factor $\rho = \rho(A_h, h, \theta)$, which measures how rapidly the linear solver converges, and the elapsed CPU time $t = t(A_h, h, \theta)$.

Then, we build our model (the ANN) to predict the performance index p of the AMG in a fixed test case and with a fixed strong threshold parameter θ , namely it is the ANN \mathcal{F} such that

$$\mathcal{F}(\mathbf{x}; \boldsymbol{\gamma}) = \mathcal{F}(\hat{V}, -\log_2(h), \theta; \boldsymbol{\gamma}) = \mathbf{y} = p(A_h, h, \theta) \quad (15)$$

where \mathbf{x} is a set of variables that identifies the matrix A_h (or better saying \hat{V} that will be obtained from A_h), h , and θ , while $\boldsymbol{\gamma}$ are the parameters that

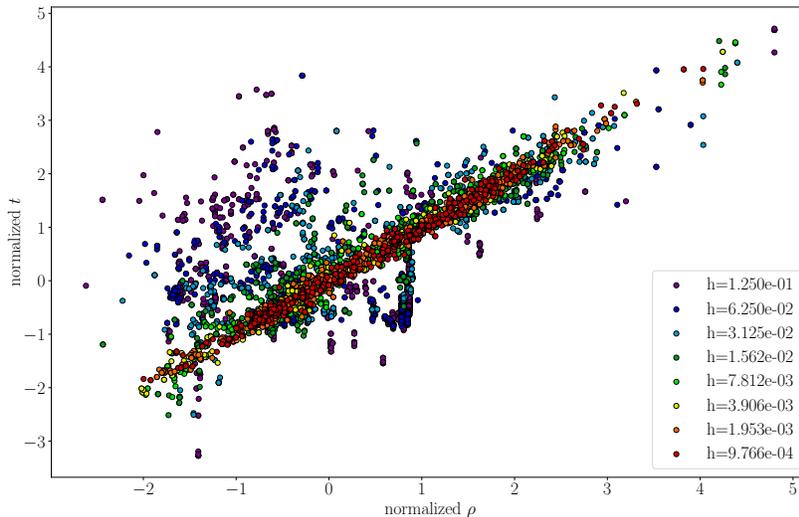


Figure 4: Scatter plot of the average elapsed CPU time (t) to solve the linear system and approximate convergence factor ρ , for different mesh sizes h . Data are normalized (in both components) with respect to the corresponding data in the same test case.

define the ANN. There are two reasons to adopt this approach: first, it is possible to quantify the improvement on the performance that we expect; second, each numerical simulation can be added to the dataset making this process less computationally expensive and more flexible.

If otherwise not stated, we will use a 60%-20%-20% split of the dataset into training-validation-test.

5.4.1 Choice of the performance index p

In order to determine the most suited measure to be employed as the performance index p for the AMG solver, we analyze the relationship between the elapsed CPU time t and the approximate convergence factor ρ . In Figure 4, we show a scattered plot of the elapsed CPU time (t) as a function of ρ , for different values of the mesh size h . We notice that the results are normalized with respect to the data that belong to the same test case. It is possible to highlight a linear relationship between t and ρ . This is also confirmed by the results shown in Table 3, where we report the least square analysis of the results of Figure 4.

We highlight that these results support the hypothesis of a relationship between ρ and t . We can explain the poor correlation for coarse mesh sizes h as due to the higher relative uncertainty of the measure. Indeed, as h gets smaller the coefficient of determination R^2 improves. Thus, we use as unique target of the ANN ρ . This choice is also motivated by the fact that ρ is not machine nor

h	1.25e-1	6.25e-2	3.12e-2	1.56e-2
Data points	1200	1200	1200	1200
R^2	0.155	0.564	0.762	0.912
Adj. R^2	0.155	0.564	0.762	0.912
F-statistic	220.5	1550	3836	1.24e+04
AIC	-2.15e+4	-1.79e+4	-1.53e+4	-1.30e+4
ρ coef.	3.49e-4	3.05e-3	1.25e-2	4.99e-2
ρ SE	2.35e-5	7.76e-5	2.02e-4	4.48e-4
ρ t-value	14.848	39.369	61.937	111.299
ρ p-value	< 0.001	< 0.001	< 0.001	< 0.001
h	7.81e-3	3.91e-3	1.95e-3	9.77e-4
Data points	1200	1200	1200	1200
R^2	0.985	0.992	0.991	0.993
Adj. R^2	0.985	0.992	0.991	0.993
F-statistic	7.88e+04	1.58e+05	1.34e+05	1.68e+05
AIC	-1.09e+4	-7864	-4048	-800.1
ρ coef.	0.263	1.245	5.259	20.95
ρ SE	9.35e-4	3.13e-3	1.42e-2	5.11e-2
ρ t-value	280.677	397.857	366.596	409.565
ρ p-value	< 0.001	< 0.001	< 0.001	< 0.001

Table 3: Least squares (LS) analysis between the elapsed CPU time t and the approximate converge factor ρ , as a function of the mesh size h . Each row from the top represents: the number of data points employed in the LS (data points); the square of the coefficient of determination (R^2); its adjusted version (Adj. R^2); the F-statistic of the regression (F-statistic); the Akaike’s information criterion (AIC); the estimated coefficient of the predictor variable (ρ coef.); its standard error (ρ SE); its t-value (ρ t-values); and its p-value ρ (p-value).

implementation dependent, thus leading to reproducible result.

5.5 ANN-based prediction of the optimal strong threshold parameter θ

In this section, we discuss how to predict the optimal strong threshold parameter θ to be used in the ANN-enhanced AMG Algorithm 3 without using any prior assumption on the diffusion coefficient μ . In other words, we do not rely on the fact that μ shows a finite number of patterns.

The variables that we use as inputs \mathbf{x} of the ANN \mathcal{F} are θ , $-\log_2(h)$ and a set of variables \hat{V} that is extracted from the matrix A_h of the linear system (9) by means of the pooling Algorithm 4 and a normalization algorithm (extraction step). This extraction process should be computationally cheap, indeed this approach is worthwhile only until the process of predicting the optimal value of θ has a negligible computational cost with respect to the elapsed CPU time to

Algorithm 4: Pooling algorithm

 $(V, C) = \text{pooling}(A_h, m)$

```
1 access  $A_h$  in COO form and extract its size:  $\text{val}, \text{row}, \text{col}, n \leftarrow A_h$ ;  
2 initialize  $V$  to an  $m \times m$  dense matrix with all zero entries;  
3 initialize  $C$  to an  $m \times m$  dense matrix with all zero entries;  
4  $q \leftarrow n/m$ ;  
5  $p \leftarrow n \bmod m$ ;  
6  $t \leftarrow (q + 1)p$ ;  
7 for  $k = 0$  to  $\text{val.size}() - 1$  do  
8    $i \leftarrow \text{row}[k] < t ? \text{row}[k]/(q + 1) : (\text{row}[k] - t)/q + p$ ;  
9    $j \leftarrow \text{col}[k] < t ? \text{col}[k]/(q + 1) : (\text{col}[k] - t)/q + p$ ;  
10   $V_{ij} \leftarrow V_{ij} + \text{val}[k]$ ;  
11   $C_{ij} \leftarrow C_{ij} + 1$ ;  
12 end  
13 return  $V, C$ ;
```

solve the linear system.

We recall at this stage that the optimal parameter θ^* to be used in the linear solver with AMG preconditioner (step 6 of Algorithm 3) is such that

$$\theta^* = \operatorname{argmin}_{\theta \in (0,1]} p \left(\mathcal{F}(\hat{V}, -\log_2(h), \theta; \gamma) \right).$$

Specifically, as we consider the performance index p to be the unique scalar output \mathbf{y} of the ANN \mathcal{F} , we can simply write:

$$\theta^* = \operatorname{argmin}_{\theta \in (0,1]} \mathcal{F}(\hat{V}, -\log_2(h), \theta; \gamma).$$

5.5.1 Pooling (step 1 of Algorithm 3)

We introduce what we call the *view* of the matrix, which is the result of the application on the pooling operator to be used in the ANNs in place of the matrix A_h . We compute an average to combine entries in the same bucket. Moreover, by accessing to a sparse matrix at a low level, we can realized the pooling at a small computational cost. For the sake of simplicity, Algorithm 4 assumes that the matrix A_h is stored in coordinate list (COO). Notice that this algorithm has complexity $O(nnz)$, where nnz is the number of non-zero elements in the matrix A_h . We have also measured the elapsed CPU time by Algorithm 4 in each simulation that we have performed: these measurements empirically show that it runs in a negligible CPU time with respect to the time needed to solve the linear system. We also notice that Algorithm 4 could easily be extended to work in parallel.

We observe that the pooling algorithm calls for setting a priori the size m of the matrix $V \in \mathbb{R}^{m \times m}$.

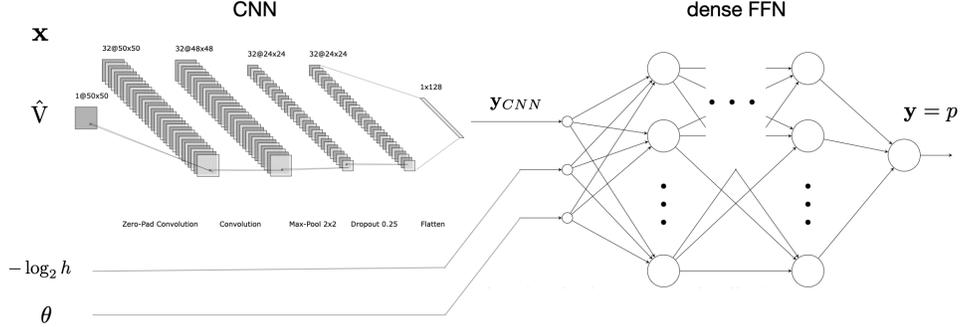


Figure 5: Architecture of the ANN $\mathbf{y} = \mathcal{F}(\mathbf{x}; \boldsymbol{\gamma})$, where $\mathbf{x} = (\hat{\mathbf{V}}(A_h), -\log_2(h), \theta)$ and $\mathbf{y} = p(A_h, h, \theta)$. The ANN is comprised by a CNN such that $\mathbf{y}_{CNN} = \mathcal{F}_{CNN}(\hat{\mathbf{V}}; \boldsymbol{\gamma}_{CNN})$ and a dense FFN $\mathbf{y} = p(A_h, h, \theta) = \mathcal{F}_{FFN}(\mathbf{y}_{CNN}, -\log_2(h), \theta; \boldsymbol{\gamma}_{FFN})$.

5.5.2 Normalization (steps 2-4 of Algorithm 3)

We observe that the *view* defined in the previous section cannot be used as input of an ANN yet. In particular, it features very large values that might impact the stability of the gradient algorithm, namely could lead to the exploding gradient problem.

For this reason, we apply a normalization to the input, namely we propose two alternatives: the standard one

$$(\hat{\mathbf{V}})_{ij} = \frac{(\mathbf{V})_{ij} - \bar{v}}{\sigma}, \quad \bar{v} = \sum_{i,j} (\mathbf{V})_{ij}, \quad \sigma = \sqrt{\frac{1}{m^2} \sum_{i,j} [(\mathbf{V})_{ij} - \bar{v}]^2}, \quad (16)$$

and the following scaled version:

$$(\hat{\mathbf{V}})_{ij} = \frac{(\mathbf{V})_{ij}}{\max_{i,j} |(\mathbf{V})_{ij}|}. \quad (17)$$

The argument behind the scaling is that we would like to preserve the sparsity pattern of the matrix. These normalization should be applied also to the mean view, that is the matrix result of the element-wise division of \mathbf{V} and \mathbf{C}

$$(\bar{\mathbf{V}})_{ij} = \frac{(\mathbf{V})_{ij}}{(\mathbf{C})_{ij}}. \quad (18)$$

Namely, we define four versions of the function $\hat{\mathbf{V}} = \text{normalize}(\mathbf{V}, \mathbf{C})$; it the composition of two maps: first we apply either Eq. (18) or $\mathbf{I}_V(\mathbf{V}, \mathbf{C}) = \mathbf{V}$, then we use either Eq. (16) or Eq. (17)

At this stage, we have obtained a matrix $\hat{\mathbf{V}} \in \mathbb{R}^{m \times m}$, with m chosen a priori. We will discuss our choice in the next section.

		Normalization Mode			
		Sum Standard	Sum Scaled	Mean Standard	Mean Scaled
Dropout 0.50	loss	1.01e-4	1.56e-4	3.43e-4	1.20e-4
	MAE	5.84e-3	7.88e-3	1.16e-2	6.40e-3
Dropout 0.25	loss	9.45e-5	1.36e-4	1.01e-4	9.97e-5
	MAE	5.47e-3	6.92e-3	5.85e-3	5.79e-3
Dropout 0.00	loss	4.26e-4	1.02e-4	1.43e-4	1.08e-4
	MAE	1.18e-2	5.98e-3	6.48e-3	5.54e-3

Table 4: Computed loss (MSE) and mean absolute error (MAE) for a model trained with different normalized input and different dropout rates.

5.5.3 ANN architecture (building the ANN of step 5 in Algorithm 3)

We now build the ANN $\mathcal{F}(\mathbf{x}; \gamma)$, whose input is $\mathbf{x} = (\hat{V}, -\log_2(h), \theta)$, while the output is the performance index $\mathbf{y} = p(A_h, h, \theta)$, which we select as the approximated convergence factor $\mathbf{y} = p = \rho(A_h, h, \theta)$. We recall that, following the former pooling and normalization steps, we have $\hat{V} = \text{normalize}(\text{pooling}(A_h))$, where `pooling` is defined in (Algorithm 4) and `normalize` is defined at the end of Section 5.5.2.

In particular, we use a model (ANN) that is the composition of two networks as depicted in Figure 5. Since the matrix view \hat{V} is a structured input, we first employ a CNN such that

$$\mathbf{y}_{CNN} = \mathcal{F}_{CNN}(\hat{V}; \gamma_{CNN}),$$

depending on the parameters γ_{CNN} . Then, the output of the first CNN altogether with the remaining inputs $-\log_2(h)$ and θ of \mathbf{x} , constitute the inputs of a second network, which we select as a dense Feed Forward Network (FFN). This dense FFN is such that

$$\mathbf{y} = p(A_h, h, \theta) = \mathcal{F}_{FFN}(\mathbf{y}_{CNN}, -\log_2(h), \theta; \gamma_{FFN})$$

and depends on the parameters γ_{CNN} .

The size m of the view, i.e. size of the matrix \hat{V} , is fixed in this paper as $m = 50$. Our choice is motivated by interpreting \hat{V} as a color image in input to the first CNN network; experience indicates that this kind of CNN network is able to excellently process color images of similar size.

Before tuning the architecture of the ANN, we compare the normalization strategies of steps 2-4 in Algorithm 3 to check which is the most effective for the ANN training. For this test, we employ a fairly simple model that is composed by 2 convolutional layers with 32 units per layer, 2×2 max-pooling, a dropout layer and 128 outputs for the CNN. The dense part is instead build as the composition of 3 layers with 64 hidden units. As shown in Table 4, the Sum Standard and Mean Scaled version consistently outperform the other two

W_1	D_1	P_1	W_2	D_2	P_2	O	W_3	D_3	loss	MAE
32	2	0.25	-	-	-	128	64	2	7.36e-5	4.33e-3
32	2	0.25	32	2	0.5	128	64	2	9.28e-5	5.40e-3
32	2	0.0	-	-	-	128	64	3	7.85e-5	4.96e-3
32	2	0.25	-	-	-	128	64	3	7.72e-5	4.94e-3
32	2	0.5	-	-	-	128	64	3	7.86e-5	5.16e-3
32	2	0.25	-	-	-	256	64	3	8.19e-5	5.13e-3
32	2	0.25	64	2	0.5	128	64	4	1.88e-4	9.68e-3

Table 5: Computer loss (MSE) and mean absolute error (MAE) for different ANNs architectures. The batch size is 32, normalization is the standard one (see (16)), training lasts 500 epochs and the optimizer is the Adam algorithm.

normalization methods. We choose using (16) from here on as it is the one that reaches the minimum loss (with dropout rate equal to 0.25).

Further results on the architecture are presented in Table 5. In particular, we consider architectures with two convolutional layers each composed by a convolution with zero-padding, 3×3 kernel and ReLU activation and $D_i - 1$ other convolutions with 3×3 kernel and ReLU activation (without padding). The last elements of the convolutional layers are a 2×2 max-pooling and Dropout with rate P_i , each layer has W_i hidden units (for $i = 1, 2$). The output of the convolutional part has O hidden units; the dense part is composed by D_3 dense layers with W_3 hidden units.

We have chosen the architecture of the first model displayed in Table 5, with training lasting up to 1000 epochs and early stopping we have obtained a loss of $6.36 \cdot 10^{-5}$. This means that the input we are using is effectively carrying the information needed to predict the approximate convergence factor since in the previous section we obtained a similar loss. Since the a priori choice of the strong threshold parameter θ is based on the map $p = \mathcal{F}(\mathbf{x}; \gamma)$, we need to verify that this prediction is accurate. For this reason, we plot in Figures 6 and 7 all the predictions. We can appreciate how the model is able to accurately capture the behaviour of the approximate convergence factor ρ .

5.5.4 Prediction capabilities

In order to further test the robustness of the model to unseen data (i.e. test cases that are not in the training set), we test the prediction capabilities of the ANN on a new dataset. We call the latter dataset, “dataset 2”, while the one employed so far is called “dataset 1”. In particular, we solve the same model problem (5) but with a different diffusion coefficient, defined as

$$\mu(x, y) = \begin{cases} 10^{\varepsilon_2} & \text{if } (x, y) \in \Omega_{gray}, \\ 10^{\varepsilon_1} & \text{if } (x, y) \in \Omega_{white}, \end{cases} \quad (19)$$

where ε_1 and ε_2 are parameters to be chosen and $\Omega_{gray}, \Omega_{white}$ is a partition of Ω as shown in Figure 1.

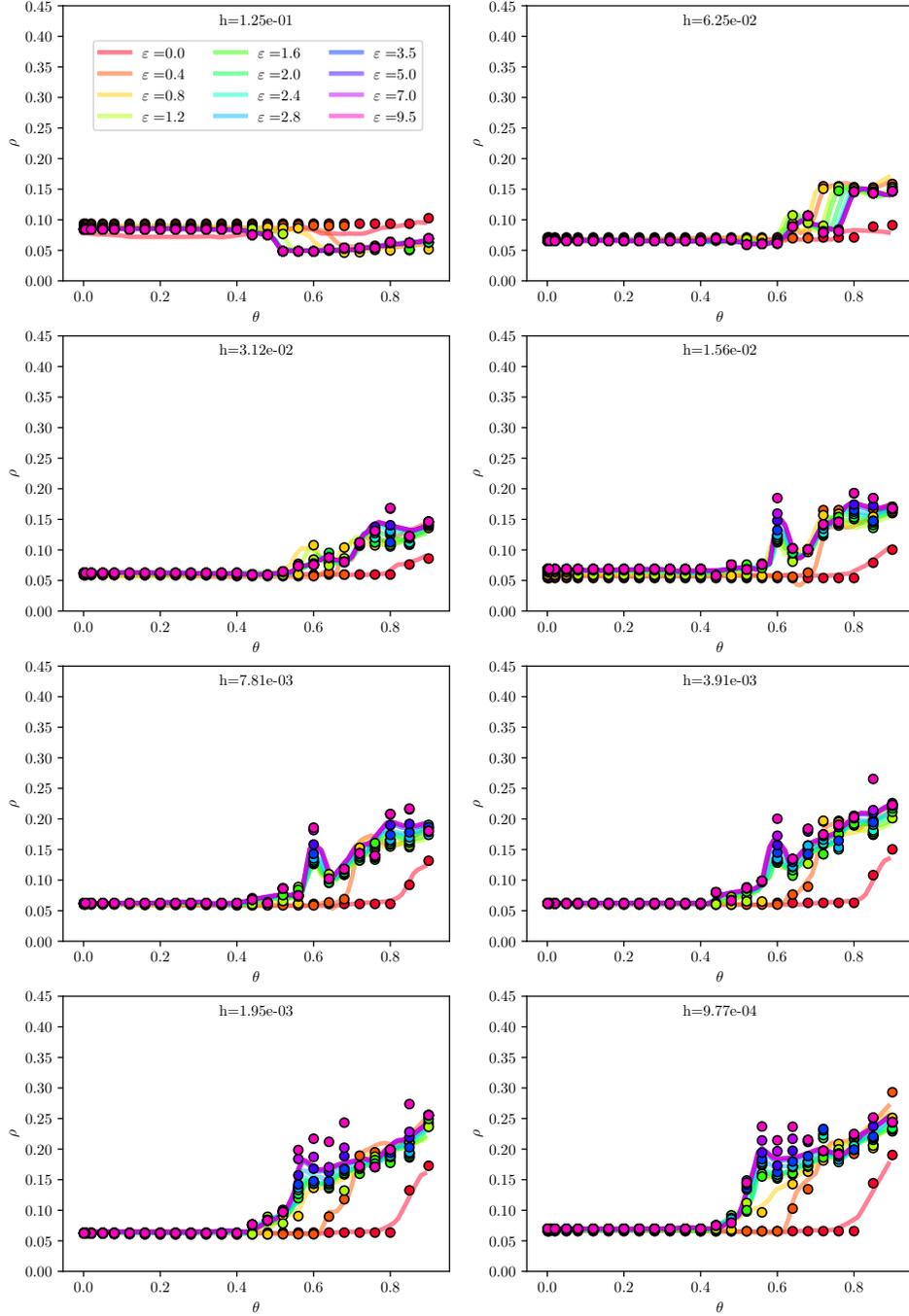


Figure 6: Predictions (line) of the computed approximate convergence factor (dots) for different values of ε . Each plot groups simulations with a different mesh size h . The pattern of μ is reported in Figure 1(c).

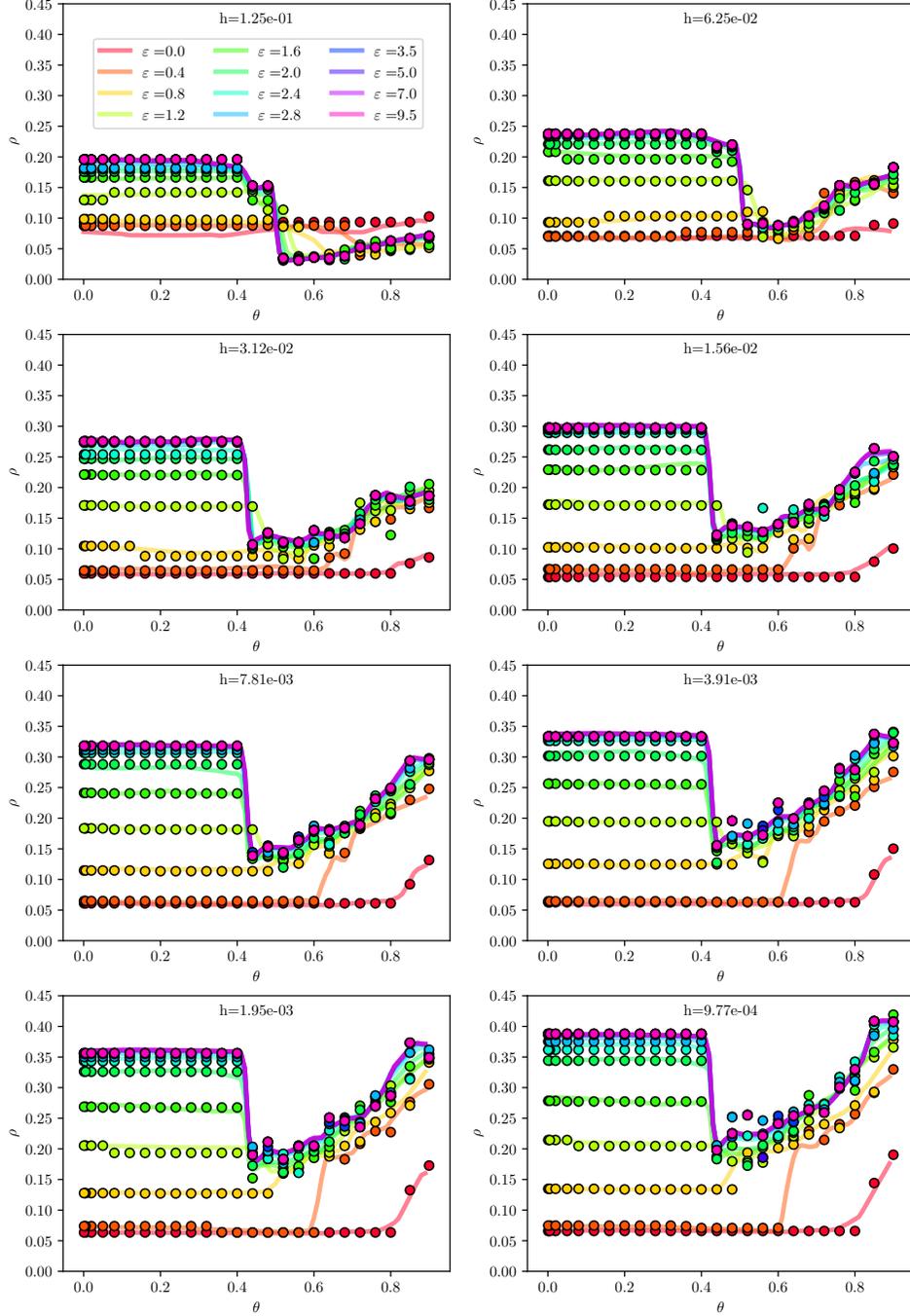


Figure 7: Predictions (line) of the computed approximate convergence factor (dots) for different values of ϵ . Each plot groups simulations with a different mesh size h . The pattern of μ is reported in Figure 1(d).

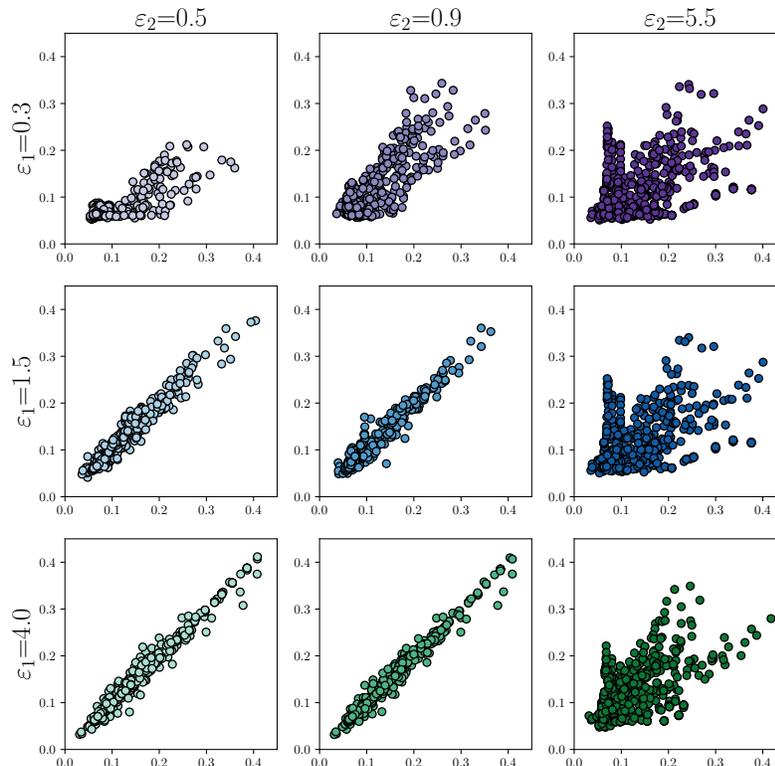


Figure 8: Predictions on dataset 2 made by the model designed in Section 5.5.3 and trained with dataset 1. Each plot has data coming from a fixed combination of ε_1 and ε_2 . Namely, ε_1 is constant for subplots on the same row and ε_2 is constant for subplots on the same column of the plot grid. On the x-axis there is the true value of ρ , on the y-axis the predicted value.

In Figure 8, we show the performances of the model that we designed in the previous sections, trained with only with dataset 1. In particular, we choose randomly three values for ε_1 and three values for ε_2 . We can see how the predictions maintain accuracy in some scenarios, but fail in other ones.

We proceed to show how the model behaves when the training is instead done with training samples from both datasets. Dataset 2 contains 5184 entries, we define the test set to be the union of the 20% of dataset 2 and the 50% of the dataset 1. In this way, the union of the training and validation set contains 4800 datapoints from the dataset 1 and 4147 from dataset 2. The ratio between the number of entries of the validation set and the training set is defined to be 1 : 3. We call this combination dataset 3. The aim is to have a balanced training dataset in which each definition of μ is equally represented.

If not otherwise stated, we stop the training at 200 epochs. As shown in

W_1	D_1	P_1	O	W_3	D_3	loss	MAE
32	2	0.05	128	64	3	1.75e-4	8.60e-3
32	2	0.00	128	64	4	1.63e-4	8.03e-3
32	2	0.25	128	64	3	1.58e-4	8.14e-3
32	2	0.25	128	64	4	1.35e-4	7.34e-3
32	2	0.50	128	64	3	1.55e-4	7.98e-3

Table 6: Computed loss (MSE) and mean absolute error (MAE) on the test “dataset 3” for different ANNs architectures with one convolutional layer. Notice how the different dropout rates affect the training.

W_1	D_1	P_1	W_2	D_2	P_2	O	W_3	D_3	loss	MAE
16	2	0.25	12	2	0.25	128	64	4	1.72e-4	8.34e-3
16	2	0.25	12	2	0.50	256	256	3	1.56e-4	8.10e-3
16	2	0.25	16	2	0.25	128	64	4	1.68e-4	8.28e-3
16	2	0.25	32	2	0.50	128	64	4	1.61e-4	8.23e-3
32	2	0.25	16	2	0.25	128	64	3	1.55e-4	8.05e-3
32	2	0.25	16	2	0.25	128	64	4	1.55e-4	7.85e-3
32	2	0.25	32	2	0.50	128	64	4	1.75e-4	8.64e-3

Table 7: Computed loss (MSE) and mean absolute error (MAE) on the test “dataset 3” for different ANNs architectures with two layers.

W_1	D_1	P_1	O	W_3	D_3	loss	MAE
32	2	0.25	128	64	3	1.71e-5	8.19e-3
32	2	0.25	128	64	4	1.35e-5	7.34e-3
32	2	0.25	128	64	5	1.48e-5	7.88e-3
32	2	0.25	128	128	3	1.51e-5	7.60e-3
32	2	0.25	128	128	4	1.43e-5	7.62e-3
32	2	0.25	256	64	5	1.50e-5	7.86e-3
32	2	0.25	256	256	3	1.52e-5	7.80e-3
32	2	0.25	512	128	3	1.60e-5	8.01e-3
32	3	0.25	128	64	4	1.51e-5	7.64e-3
32	3	0.25	128	128	3	1.48e-5	7.79e-3
32	3	0.25	128	128	4	1.34e-5	7.18e-3
32	3	0.25	256	256	3	1.56e-5	8.10e-3

Table 8: Computed loss (MSE) and mean absolute error (MAE) on the test “dataset 3” for different ANNs architectures with one convolutional layer. In this casem we change only the hyperparameters of the dense layers.

W_1	D_1	P_1	O	W_3	D_3	loss	MAE
16	4	0.25	128	128	4	1.32e-4	7.29e-3
16	4	0.50	128	128	4	1.56e-4	7.98e-3
16	5	0.25	128	128	4	1.54e-4	7.77e-3
16	3	0.50	128	128	4	1.51e-4	7.86e-3
24	2	0.25	128	128	4	1.53e-4	7.71e-3
24	2	0.50	128	128	4	1.52e-4	7.76e-3
24	3	0.25	128	128	4	1.40e-4	7.34e-3
24	4	0.50	128	128	4	1.60e-4	7.98e-3
32	3	0.25	128	128	4	1.34e-5	7.18e-3
32	3	0.50	128	128	4	1.47e-5	7.75e-3
40	2	0.25	128	128	4	1.27e-4	7.30e-3
40	3	0.25	128	128	4	1.32e-4	7.18e-3

Table 9: Computed loss (MSE) and mean absolute error (MAE) on the test “dataset 3” for different ANNs architectures with one convolutional layer. In this table we change only the hyperparameters of the convolutional layer.

Table 6, dropout improves the training, thus it will be employed in all the models. We have also tried employing batch normalization as a regularization technique on some of these models and a deeper model with three convolutional layers but it did not lead to any significant improvements. This can be explained by the fact that batch normalization effectiveness is most evident in very deep models; see [18].

Table 7 shows training of models with two convolutional layers. By comparing it with Table 8, where the MSE and MAE are reported for different ANN architectures with one layer, it is possible to notice that models with only one layer achieve lower loss. From Table 8, it is also possible to appreciate that the model that in the previous section achieved the lowest loss is not the same in this case. In particular, a deeper model performs better. This is not surprising since this means that we need a more complex model to explain the data, and indeed we are using a more diversified dataset. In Table 9, we repeat the same test case for different architectures of the convolutional layer. Two applications of convolution with 40 hidden units seems to be the best choice. The architecture that we choose for the model is the second to last of Table 9. Employing training with batch size 32, the Adam optimizer and early stopping (up to 1000 epochs), we obtain a loss on the test “dataset 3” of $8.83 \cdot 10^{-5}$ and MAE $4.84 \cdot 10^{-3}$. On the test “dataset 1” we achieve a loss of $7.73 \cdot 10^{-5}$ and a MAE of $4.48 \cdot 10^{-3}$ and on test “dataset 2” we obtained a loss of $1.39 \cdot 10^{-4}$ and a MAE of $6.51 \cdot 10^{-3}$. The predictions are reported in Figure 9. It is possible to appreciate that the model seems to be accurate.

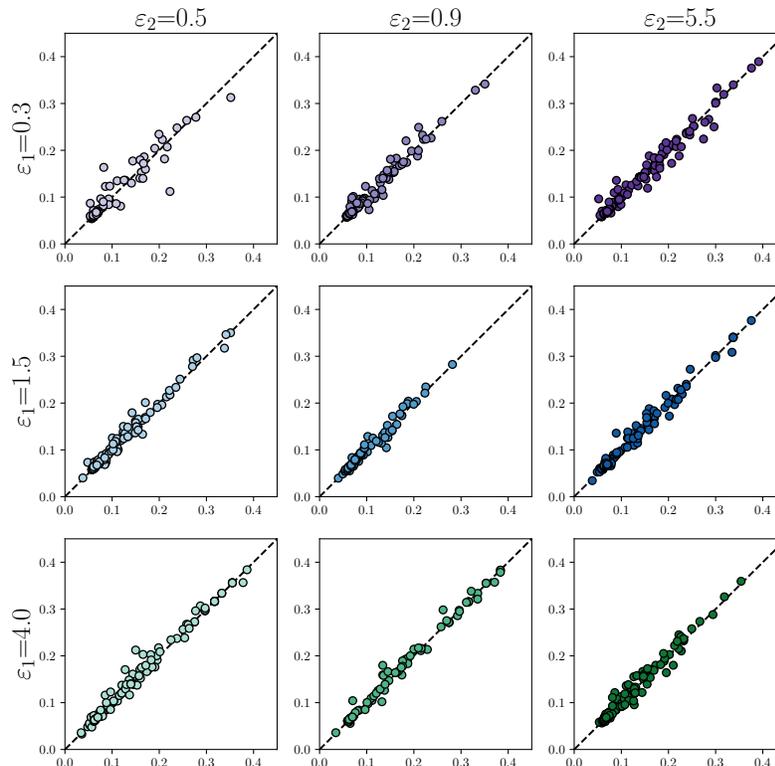


Figure 9: Prediction of the model trained with dataset 3 on the test dataset 2. Each plot has data coming from a fixed combination of ε_1 and ε_2 . Namely, ε_1 is constant for subplots on the same row and ε_2 is constant for subplots on the same column of the plot grid. On the x-axis there is the true value of ρ , on the y-axis the predicted value.

6 Conclusions

In this work, we developed an ANN-based approach to enhance the computational efficiency of the AMG methods, i.e. to accelerate their performances. In particular, we accurately predicted the value of the strong threshold parameter θ that maximizes the performance with respect to the matrix A_h of the linear system to be solved. In order to be able apply the model independently of the matrix of the linear system, we introduced a pooling operator. We measured the efficiency of the AMG method using the approximate convergence factor and we designed a model that predicts its value. In this way, we are able to choose the strong threshold parameter that minimizes the predicted approximated convergence factor. Moreover, we have shown that, as expected, the approximated convergence factor of the AMG method is strictly correlated to the elapsed CPU

time during the application of the AMG method to the linear system solution, thus demonstrating that it provides a good measure of the performances of the solver. This a priori, optimal selection of the strong threshold parameter allows us to efficiently find a value of θ that significantly decreases the elapsed CPU time with respect to the “classical” value. This improvement can be up to 33%.

Possible further developments include: using the ANN to optimize the value of other AMG parameters; testing different pooling strategies, for instance reducing the bins via the `max` function; testing the impact of varying the size of the view; testing more meaningful problems in Engineering and Applied Sciences, in particular in three-dimensional domains.

References

- [1] P. F. Antonietti and E. Manuzzi. Refinement of polygonal grids using Convolutional Neural Networks with applications to polygonal Discontinuous Galerkin and Virtual Element methods. *arXiv preprint arXiv:2102.05738*, 2021.
- [2] P. F. Antonietti and L. Melas. Algebraic multigrid schemes for high-order nodal discontinuous Galerkin methods. *SIAM Journal on Scientific Computing*, 42(2):A1147–A1173, 2020.
- [3] S. F. Ashby and R. D. Falgout. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. *Nuclear Science and Engineering*, 124(1):145–159, 1996.
- [4] A. H. Baker, T. V. Kolev, and U. M. Yang. Improving algebraic multigrid interpolation operators for linear elasticity problems. *Numerical Linear Algebra with Applications*, 17(2-3):495–517, 2010.
- [5] R. Bank, R. Falgout, T. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge. Algebraic multigrid domain and range decomposition (amg-dd/amg-rd). *SIAM Journal on Scientific Computing*, 37(5):S113–S136, 2015.
- [6] P. Bastian, M. Blatt, and R. Scheichl. Algebraic multigrid for discontinuous Galerkin discretizations of heterogeneous elliptic problems. *Numerical Linear Algebra with Applications*, 19(2):367–388, 2012.
- [7] A. Brandt and O. E. Livne. *Multigrid Techniques*. Society for Industrial and Applied Mathematics, 2011.
- [8] J. Brannick, M. Brezina, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge. An energy-based amg coarsening strategy. *Numerical Linear Algebra with Applications*, 13(2-3):133–148, 2006.
- [9] M. Brezina, A. Cleary, R. Falgout, V. Henson, J. Jones, T. Manteuffel, S. McCormick, and J. Ruge. Algebraic multigrid based on element interpolation (amge). *SIAM Journal on Scientific Computing*, 22, 2002.

- [10] H. Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Springer Science & Business Media, 2010.
- [11] Q. M. Bui, L. Wang, and D. Osei-Kuffuor. Algebraic multigrid preconditioners for two-phase flow in porous media with phase transitions. *Advances in Water Resources*, 114:19–28, 2018.
- [12] N. Discacciati, J. S. Hesthaven, and D. Ray. Controlling oscillations in high-order discontinuous Galerkin schemes using artificial viscosity tuned by neural networks. *Journal of Computational Physics*, 409:109304, 2020.
- [13] R. Falgout and S. Vassilevski. On generalizing the amg framework. *SIAM Journal on Scientific Computing*, 42(4):1669–1693, 2004.
- [14] R. Falgout and U. Yang. hypre: A library of high performance preconditioners. *Computational Science-ICCS 2002, Pt Iii, Proceedings*, 2331:632–641, 04 2002.
- [15] S. Fresca, L. Dede', and A. Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *Journal of Scientific Computing*, 87(2):1–36, 2021.
- [16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.
- [17] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [19] A. Heinlein, A. Klawonn, M. Lanser, and J. Weber. Combining machine learning and adaptive coarse spaces-a hybrid approach for robust FETI-DP methods in three dimensions. *SIAM Journal on Scientific Computing*, 0(0):S816–S838, 2021.
- [20] V. E. Henson and P. S. Vassilevski. Algebraic multigrid preconditioners for two-phase flow in porous media with phase transitions. *SIAM Journal on Scientific Computing*, 23(2):629–650, 2001.
- [21] J. S. Hesthaven and S. Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, 2018.
- [22] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

- [23] T. J. R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Courier Corporation, 2012.
- [24] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [25] M. Janssens and S. Hulshoff. Advancing artificial neural network parameterisation for atmospheric turbulence using a variational multiscale model. *Journal of Advances in Modeling Earth Systems*, page e2021MS002490, 2021.
- [26] J. Jones and B. Lee. A multigrid method for variable coefficient maxwell’s equations. *SIAM Journal on Scientific Computing*, 27(5):1689–1708, 2006.
- [27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [28] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.
- [29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [30] J. L. Lions and E. Magenes. *Non-Homogeneous Boundary Value Problems and Applications*. Springer-Verlag Berlin Heidelberg, 1972.
- [31] S. Mishra. A machine learning framework for data driven acceleration of computations of differential equations. *Mathematics in Engineering*, 1(1):118–146, 2019.
- [32] P. Neittaanmäki and S. Repin. Artificial intelligence and computational science. *Intelligent Systems, Control and Automation: Science and Engineering*, 76:27–35, 2022.
- [33] A. Quarteroni. *Numerical Models for Differential Problems*, volume 16. Springer International Publishing, 2017.
- [34] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations*. Springer-Verlag, Berlin and Heidelberg, 1994.
- [35] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Machine learning of linear differential equations using Gaussian processes. *Journal of Computational Physics*, 348:683–693, 2017.
- [36] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

- [37] F. Regazzoni, L. Dede', and A. Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. *Journal of Computational Physics*, 397:108852, 2019.
- [38] J. W. Ruge and K. Stüben. 4. *Algebraic Multigrid*, pages 73–130. 1987.
- [39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
- [40] C. Siefert, R. Tuminaro, A. Gerstenberger, G. Scovazzi, and S. S. Collis. Algebraic multigrid techniques for discontinuous Galerkin methods with varying polynomial order. *Computational Geosciences*, 18(5):597–612, 2014.
- [41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [42] K. Stüben. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, 128(1):281–309, 2001. Numerical Analysis 2000. Vol. VII: Partial Differential Equations.
- [43] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.
- [44] T. Tassi, A. Zingaro, and L. Dede'. A Machine Learning approach to enhance the SUPG stabilization method for advection-dominated differential problems. *MOX Report, Politecnico di Milano*, 58, 2021.
- [45] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.
- [46] R. Webster. An algebraic multigrid solver for Navier-Stokes problems. *International Journal for Numerical Methods in Fluids*, 18(8):761–780, 1994.
- [47] J. Xu and L. Zikatanov. Algebraic multigrid methods. *Acta Numerica*, 26:591–721, 2017.

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

- 66/2021** Antonietti, P.F.; Botti, M.; Mazzieri, I.
On mathematical and numerical modelling of multiphysics wave propagation with polygonal Discontinuous Galerkin methods
- 65/2021** Mazzieri, I.; Muhr, M.; Stupazzini, M.; Wohlmuth, B.
Elasto-acoustic modelling and simulation for the seismic response of structures: The case of the Tahtali dam in the 2020 Izmir earthquake
- 67/2021** Salvador, M.; Regazzoni, F.; Pagani, S.; Dede', L.; Trayanova, N.; Quarteroni, A.
The role of mechano-electric feedbacks and hemodynamic coupling in scar-related ventricular tachycardia
- 68/2021** Regazzoni, F.; Salvador, M.; Dede', L.; Quarteroni, A.
A machine learning method for real-time numerical simulations of cardiac electromechanics
- 64/2021** Clarotto, L.; Allard, D.; Menafoglio, A.
A new class of alpha-transformations for the spatial analysis of Compositional Data
- 63/2021** Rosafalco, L.; Torzoni, M.; Manzoni, A.; Mariani, S.; Corigliano, A.
Online structural health monitoring by model order reduction and deep learning algorithms
- 62/2021** Lupo Pasini, M.; Burcul, M.; Reeve, S.; Eisenbach, M.; Perotto, S.
Fast and accurate predictions of total energy for solid solution alloys with graph convolutional neural networks
- 60/2021** Rosafalco, L.; Manzoni, A.; Mariani, S.; Corigliano, A.
Fully convolutional networks for structural health monitoring through multivariate time series classification
- 61/2021** Buchwald, S.; Ciaramella, G.; Salomon, J.; Sugny, D.
A greedy reconstruction algorithm for the identification of spin distribution
- 59/2021** Stella, S.; Regazzoni, F.; Vergara, C.; Dede', L.; Quarteroni, A.
A fast cardiac electromechanics model coupling the Eikonal and the nonlinear mechanics equations