Improving pseudo-time stepping convergence for CFD simulations with neural networks

Zandbergen, Anouk; Noorden, Tycho van; Heinlein, Alexander

DOI
10.1016/j.camwa.2025.07.006

Publication date
2025

Document Version
Final published version

Published in
Computers & Mathematics with Applications

Computers and Mathematics with Applications 196 (2025) 64–83

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Improving pseudo-time stepping convergence for CFD simulations with neural networks

Anouk Zandbergen [a,b], Tycho van Noorden [b], Alexander Heinlein [c,*]

[a] Chair of Cyber-Physical Systems in Mechanical Engineering, Technische Universität Berlin, Straße des 17. Juni 135, Berlin, 10623, Germany
[b] COMSOL BV, Röntgenlaan 37, Zoetermeer, 2719 DX, the Netherlands
[c] Delft Institute of Applied Mathematics, Delft University of Technology, Mekelweg 4, Delft, 2628 CD, the Netherlands

ARTICLE INFO

Dataset link: https://github.com/searhein/nn-pseudo-time-stepping-data

MSC:
35Q30
65H10
65N12
65N22
68T07

Keywords:
Computational fluid dynamics
Newton's method
Pseudo-time stepping
Neural network
Hybrid algorithm

ABSTRACT

Computational fluid dynamics (CFD) simulations of viscous fluids described by the stationary Navier–Stokes equations are considered. Depending on the Reynolds number of the flow, the Navier–Stokes equations may exhibit a highly nonlinear behavior. The system of nonlinear equations resulting from the discretization of the Navier–Stokes equations can be solved using nonlinear iteration methods, such as Newton's method. However, fast quadratic convergence is typically only obtained in a local neighborhood of the solution, and for many configurations, the classical Newton iteration does not converge at all. In such cases, so-called globalization techniques may help to improve convergence.

In this paper, pseudo-time stepping (also known as pseudo-transient continuation) is employed in order to improve nonlinear convergence. The classical algorithm is enhanced by a neural network model that is trained to predict a local pseudo-time step. Generalization of the novel approach is facilitated by predicting the local pseudo-time step separately on each element using only local information on a patch of adjacent elements as input. Numerical results for standard benchmark problems, including flow over a backward facing step geometry and Couette flow, show the performance of the machine learning-enhanced globalization approach; as the software for the simulations, the CFD Module of COMSOL Multiphysics® is employed.

1. Introduction

Computational fluid dynamics (CFD) simulations are highly relevant for a wide range of applications, including the fields of aerospace, environmental and biological engineering, weather predictions and medicine. Numerical simulations of Newtonian fluids involve the solution of the Navier–Stokes equations. Depending on the flow regime, the Navier–Stokes equations exhibit strong nonlinearities, posing challenges to numerical solvers.

In this work, we consider two-dimensional stationary CFD simulations using a mixed finite element method (FEM). The resulting discretized nonlinear system of equations is then solved using Newton's method. Newton's method converges quadratically for initial guesses close enough to the solution, but it may converge slowly or even diverge for guesses further away. To improve global convergence of the method, globalization techniques have been developed; see [1] for an overview, in which the authors categorize

* Corresponding author.
E-mail addresses: anouk.zandbergen@tu-berlin.de (A. Zandbergen), tycho.vannoorden@comsol.com (T. van Noorden), a.heinlein@tudelft.nl (A. Heinlein).
URL: https://searhein.github.io/ (A. Heinlein).

https://doi.org/10.1016/j.camwa.2025.07.006
Received 4 July 2024; Received in revised form 26 May 2025; Accepted 4 July 2025

Available online 16 July 2025

the techniques into *backtracking methods* and *trust region methods*; cf. [2,3]. Other examples of robustness improving methods include homotopy, continuation, pseudo transient continuation, mesh sequencing methods, or nonlinear preconditioning methods; cf. [4–13].

The combination of scientific computing and machine learning, also called scientific machine learning (SciML), is a new field [14], which has recently gained a lot of attention. A popular application area for SciML techniques is CFD simulations; see the review paper [15]. Examples of successful SciML techniques in CFD are: discretization approaches using neural networks (NNs) [16–18], surrogate models [19–22], and model discovery [23–25]. In [26], the temporal evolution of dynamic systems is learned using NNs with long-short-term memory (LSTM). Approaches enhancing CFD simulations via ML include the generation of optimized meshes [27], the improvement of the resolution of the simulation results [28–31], or the detection of troubled cells in simulation meshes due to Gibbs oscillations arising from the use of higher-order methods [32]. We note that, in [29–32], the network inputs and output are specifically chosen from local patches of elements to achieve generalizability of the ML model; we apply a similar approach in this work.

The use of ML models to enhance numerical solvers is investigated in several works. In the context of linear solvers, [33] and [34] employ features generated by proper orthogonal decomposition (POD) to develop a modified conjugate gradient (CG) solver or preconditioning techniques, respectively. Enhancing the CG method by a nonlinear convolutional NN-based preconditioner is investigated in [35]. In [36,37], NNs are used to improve domain decomposition methods, and in [38–42] components of multigrid methods are selected or generated using ML models. In [43–47], suitable numerical algorithms are selected using ML techniques. Related to that, [48] investigate the prediction of a suitable combination of preconditioner and iterative method for sparse linear systems.

Fewer approaches have been developed for improving nonlinear convergence using ML. In hybrid Newton methods, the initial guess of the Newton iteration is predicted using ML models, such as NNs or random forests; see, e.g. [49–51]. In [52], the convergence of solvers for nonlinear partial differential equations is improved using POD features in a preconditioning approach, and the proposed method is tested specifically for CFD problems. In [53], dimensionless numbers and simulation properties are used as input to a random forest model to predict a relaxation parameter to accelerate convergence of the Picard iteration for multiphase porous media flow.

To the best of the authors' knowledge, this work is the first to address the improvement of the convergence of pseudo-time stepping [8] for CFD simulations using a NN model. Moreover, the approach presented here stands out due to the strong locality of the inputs and outputs. This approach is only previously described in the master thesis of the first author [54], laying the foundation for this paper. Similar to previous approaches [29–32], the used NN model is trained to predict the local pseudo-time step size for each mesh element using only local input features; this aims to ensure generalizability of the approach. We test the performance and generalizability of our approach using the FEM software package COMSOL Multiphysics®; as a reference, we compare our approach against two default pseudo-time step control mechanisms implemented in COMSOL Multiphysics®. As the ML framework, we employ Matlab, which enables us to access data from COMSOL via an available interface between the two packages.

The paper is organized as follows. First, the Navier-Stokes equations and pseudo-time stepping are briefly recalled and discussed in section 2 respectively section 3. In section 4 the NN for the local element pseudo-time step prediction is described. Numerical results are given in section 5. Lastly, conclusions are presented in section 6.

2. Navier–Stokes equations

The Navier–Stokes equations are a system of partial differential equations (PDEs) that describes the flow of Newtonian fluids, that is, fluids with a linear correlation of the viscous stresses and the local strain rate. In particular, we consider incompressible fluids, that is, fluids with a constant density; cf. [55]. The stationary Navier-Stokes equations for incompressible flow read

$$\begin{aligned} \rho(\mathbf{u}\cdot\nabla)\mathbf{u} - \mu\nabla^2\mathbf{u} &= -\nabla p + \mathbf{f}, \\ \rho\nabla\cdot\mathbf{u} &= 0, \end{aligned} \tag{1}$$

describing the conservation of momentum and mass, respectively. In these equations, $\mathbf{u}$ is the flow velocity, μ is the dynamic viscosity of the fluid, ρ is the density, which is assumed to be constant, p is the pressure, and $\mathbf{f}$ is the body force acting on the flow [55]; in all our experiments, the body force is assumed to be zero.

Due to the convective term $\rho(\mathbf{u}\cdot\nabla)\mathbf{u}$, eq. (1) is nonlinear, and the numerical treatment of this nonlinearity is the subject of this research. Very strong nonlinearities are generally related to turbulent flow. However, already in the regime of laminar flow, the nonlinearities may become severe enough to cause nonlinear solvers to diverge; hence, we will focus on the laminar case. Flow remains laminar as long as the Reynolds number is below a critical value in the order of 10^5; see, e.g., [56]. The Reynolds number Re is given by

$$\mathrm{Re} = \frac{\rho U L}{\mu}. \tag{2}$$

Here, U and L are the typical velocity and length scale. The Reynolds number describes the ratio between inertial and viscous forces; cf. [57]. At low Reynolds numbers, the flow remains laminar as the viscous forces are able to damp out disturbances in the flow. At high Reynolds numbers, the inertial forces become more important resulting in nonlinear interactions to grow, which causes turbulence.

In order to solve the stationary Navier-Stokes equations in eq. (1) on a Lipschitz domain $\Omega \subset \mathbb{R}^2$ numerically, we consider a triangulation of Ω into linear triangles. Then, we discretize eq. (1) using piecewise linear finite elements for both the velocity $\mathbf{u}$ and the pressure p. Since this type of discretization does not satisfy the inf-sup condition, Galerkin least-squares (GLS) streamline diffusion is employed to stabilize the discretized problem. Furthermore, Hughes–Mallet type shock-capturing is used for additional robustness

in case of sharp gradients, either internally or at boundaries. For more details on the employed stabilization techniques and the used stabilization parameters, see [58,59] and the references therein. This yields a discrete but nonlinear system of equations

$$\begin{aligned} N(\mathbf{u}_h) + B^\top p_h &= 0, \\ B\mathbf{u}_h &= 0, \end{aligned} \tag{3}$$

where we denote the discrete velocity and pressure fields as $\mathbf{u}_h$ respectively p_h. The linear operators B and $B^\top$ correspond to the divergence and gradient, respectively, and the nonlinear operator N corresponds to the convection-diffusion of the velocity field.

By combining the fields $\mathbf{u}_h$ and p_h into a single field $\mathbf{v}$, we can simply formulate the discrete nonlinear problem eq. (3) as

$$F(\mathbf{v}) = 0.$$

We solve this system using Newton's method, that is, by solving a sequence of linear systems of the form

$$-F'(\mathbf{v}^n)\Delta\mathbf{v}^n = F(\mathbf{v}^n),$$

where n is the iteration index of the Newton iteration and $F'(\mathbf{v}^n)$ is the Jacobian at the linearization point $\mathbf{v}^n$. Close to the solution, Newton's method converges with quadratic rate. However, for an arbitrary initial guess, the method might also diverge. As we will see in our numerical results, this may happen for a wide range of cases for the stationary Navier–Stokes equations. In order to make the nonlinear iteration more robust, globalization techniques can be employed; see, e.g., [1] for an overview of globalization techniques for Newton's method for the Navier–Stokes equations. Here, we will consider the pseudo-time stepping approach [8], which we will discuss in more detail in the next section.

3. Pseudo-time stepping

Solving stationary nonlinear equations using Newton's method requires an initial guess close enough to the root. Globalization techniques are designed to give a result for a wider range of initial guesses, but can stagnate at local minima [8]. Pseudo-time stepping, or pseudo-transient continuation, is an alternative to Newton's method for computing stationary solutions of time-dependent partial differential equations. The idea is to obtain a solution more robustly by solving the time-dependent equation instead of the stationary one. The idea behind the method is to frame the problem in a time-depending setting,

$$\frac{\partial \mathbf{v}}{\partial t} = -F(\mathbf{v}), \tag{4}$$

with $F(\mathbf{v}) = 0$ the system of nonlinear equations of which a solution must be determined. Now the algorithm for pseudo-time stepping can be described as the numerical integration with a variable time step method of the initial value problem

$$\frac{\partial \mathbf{v}}{\partial t} = -F(\mathbf{v}), \quad \mathbf{v}(0) = \mathbf{v}_0; \tag{5}$$

cf. [8]. The idea is that this time integration converges to a steady state for a wider range of initial values $\mathbf{v}_0$ than the standard Newton method. In addition, the method tries to increase the time step as $F(\mathbf{v})$ approaches 0, such that when the iterations get closer to a steady state, the convergence becomes (close to) quadratic.

The iterations of the pseudo-time stepping algorithm are given by

$$\mathbf{v}^{n+1} = \mathbf{v}^n - \left((\Delta t^n)^{-1} I + F'(\mathbf{v}^n)\right)^{-1} F(\mathbf{v}^n), \tag{6}$$

with $F'(\mathbf{v}^n)$ the Jacobian. To understand how eq. (6) is obtained from eq. (5) consider an Euler backward step starting from $\mathbf{v}^n$ for eq. (5):

$$\mathbf{z}^{n+1} = \mathbf{v}^n - \Delta t^n F(\mathbf{z}^{n+1}). \tag{7}$$

Note that $\mathbf{z}^{n+1}$ is a root of

$$G(\xi) := \xi + \Delta t^n F(\xi) - \mathbf{v}^n, \tag{8}$$

as a function of ξ. The first Newton iterate for solving $G(\xi) = 0$ with initial guess $\xi_0 = \mathbf{v}^n$ gives

$$\begin{aligned} \xi_1 &= \mathbf{v}^n - \left(I + \Delta t^n F'(\mathbf{v}^n)\right)^{-1} (\mathbf{v}^n + \Delta t^n F(\mathbf{v}^n) - \mathbf{v}^n), \\ &= \mathbf{v}^n - \left((\Delta t^n)^{-1} + F'(\mathbf{v}^n)\right)^{-1} F(\mathbf{v}^n). \end{aligned}$$

From this corrector iteration, we have obtained the formula in eq. (6). The pseudo code of the method is the given by Algorithm 1.

3.1. Applying pseudo-time stepping to the weak form of the Navier-Stokes equations

Consider a Lipschitz domain $\Omega \in \mathbb{R}^2$ with boundary $\Gamma = \Gamma_D \cup \Gamma_N$. For the weak formulation of the Navier-Stokes equations, we introduce the following functions spaces:

Algorithm 1 Pseudo-time stepping algorithm from [8].

1: Set $\mathbf{v} = \mathbf{v}_0$ and $\Delta t = \Delta t_0$.
2: **while** $\|F(\mathbf{v})\|$ is too large **do**
3: Solve $(\Delta t^{-1} I + F'(\mathbf{v}))\mathbf{s} = -F(\mathbf{v})$
4: Set $\mathbf{v} = \mathbf{v} + \mathbf{s}$
5: Evaluate $F(\mathbf{v})$
6: Update Δt
7: **end while**

$$L^2(\Omega) = \{v : \Omega \to \mathbb{R} | \int v^2 \, dx < \infty\},$$

$$H^1(\Omega) = \{v : \Omega \to \mathbb{R} | \int (v^2 + |\nabla v|^2) \, dx < \infty\},$$

$$V = \{\boldsymbol{v} \in (H^1(\Omega))^2\},$$

$$V_0 = \{\boldsymbol{v} \in (H^1(\Omega))^2 | \boldsymbol{v} = 0 \text{ on } \Gamma_D\}.$$

The weak form of the Navier-Stokes equations is given by: find $\mathbf{u} \in L^2(0,T;V) \cap C^0(0,T;L^2(\Omega))$ and $p \in L^2((0,T)\times\Omega)$ such that

$$\begin{aligned} \int_\Omega (\rho \partial_t \mathbf{u} + \rho(\mathbf{u}\cdot\nabla)\mathbf{u}) \cdot \phi - (\mu \nabla \mathbf{u} \cdot \nabla \phi - p \nabla \cdot \phi) \, dx &= \int_\Omega \mathbf{f} \cdot \phi \, dx, \\ \int_\Omega \rho \nabla \cdot \mathbf{u} \psi \, dx &= 0, \end{aligned} \tag{9}$$

for all test functions $\phi \in V_0$, $\psi \in L^2(\Omega)$, and $\mathbf{u}$ satisfying the boundary condition $\mathbf{u}|_{\Gamma_D} = g$ and initial condition $\mathbf{u}|_{t=0} = \mathbf{u}_0$. Here, $\mathbf{f}$ is a forcing term. Note that this formulation is derived assuming that, on the Neumann boundary Γ_N, the velocity $\mathbf{u}$ and pressure p satisfy

$$\mu \mathbf{n} \cdot \nabla \mathbf{u} - \mathbf{n} p = 0,$$

with $\mathbf{n}$ the outer normal of the boundary Γ.

Next, we discretize the equations in space using finite elements. Therefore, let $\mathcal{T}_h$ be a triangulation of the domain Ω with maximum element size h. For every element $\mathcal{T} \in \mathcal{T}_h$, let $P_1(\mathcal{T})$ denote the set consisting of all linear functions on $\mathcal{T}$. Associated with $\mathcal{T}_h$, we define the finite element spaces

$$X_h = \{\mathbf{u} \in C(\Omega)^2 : \mathbf{u}|_{\mathcal{T}} \in P_1(\mathcal{T})^2, \forall \mathcal{T} \in \mathcal{T}_h, \mathbf{u}|_{\Gamma_D} = g\},$$

$$X_{0h} = \{\mathbf{u} \in C(\Omega)^2 \cap V_0 : \mathbf{u}|_{\mathcal{T}} \in P_1(\mathcal{T})^2, \forall \mathcal{T} \in \mathcal{T}_h\},$$

$$Y_h = \{p \in C(\Omega) : p|_{\mathcal{T}} \in P_1(\mathcal{T}), \forall \mathcal{T} \in \mathcal{T}_h\},$$

which will replace the infinite dimensional function spaces in the weak formulation.

Then, in order to apply pseudo-time stepping to the stationary form of the spatially discretized system, the same conceptual idea as described in the previous section is used, which can be summarized by first writing the time-discretized equations for one Euler backward step, and then linearizing this system for the new time step around the current time step. The Euler backward formula with time step Δt^n for the discretized weak form of eq. (9) gives the following (discretized) nonlinear problem: given $(\mathbf{u}_h^n, p_h^n) \in X_h \times Y_h$ find $(\mathbf{u}_h^{n+1}, p_h^{n+1}) \in X_h \times Y_h$ such that

$$\begin{aligned} \sum_{\mathcal{T}\in\mathcal{T}_h} \Bigg(\int_{\mathcal{T}} (\rho \frac{\mathbf{u}_h^{n+1} - \mathbf{u}_h^n}{\Delta t^n} + \rho(\mathbf{u}_h^{n+1}\cdot\nabla)\mathbf{u}_h^{n+1}) \cdot \phi_j \\ -(\mu \nabla \mathbf{u}_h^{n+1} \cdot \nabla \phi_j - p_h^{n+1} \nabla \cdot \phi_j) \, dx - \int_{\mathcal{T}} \mathbf{f} \cdot \phi_j \, dx \Bigg) = 0, \\ \sum_{\mathcal{T}\in\mathcal{T}_h} \left(\int_{\mathcal{T}} \rho \nabla \cdot \mathbf{u}_h^{n+1} \psi_j \, dx \right) = 0, \end{aligned} \tag{10}$$

for all test functions $(\phi_j, \psi_j) \in X_{0h} \times Y_h$. Note that, in order to make this discretization stable, additional stabilization terms are needed. The standard choice for these stabilization terms in COMSOL Multiphysics is to employ Galerkin least-squares (GLS) streamline diffusion in combination with a Hughes–Mallet type shock-capturing for robustness in case of sharp gradients. For more details, see [58,59]. Here, these terms are omitted for the ease of presentation as they are not essential for the explanation of the main ideas.

Linearizing this system at $(\mathbf{u}_h^n, p_h^n)$ results in

$$\begin{aligned}\sum_{\mathcal{T}\in\mathcal{T}_h}\Bigg(\int_{\mathcal{T}}(\rho\frac{\mathbf{u}_h^{n+1}-\mathbf{u}_h^n}{\Delta t^n}+\rho(\mathbf{u}_h^n\cdot\nabla)\mathbf{u}_h^{n+1}+\rho(\mathbf{u}_h^{n+1}\cdot\nabla)\mathbf{u}_h^n\\ -\rho(\mathbf{u}_h^n\cdot\nabla)\mathbf{u}_h^n)\phi_j-\int_{\mathcal{T}}(\mu\nabla\mathbf{u}_h^{n+1}\cdot\nabla\phi_j-p_h^{n+1}\nabla\cdot\phi_j)\,dx-\int_{\mathcal{T}}\mathbf{f}\cdot\phi_j\,dx\Bigg)=0,\\ \sum_{\mathcal{T}\in\mathcal{T}_h}\Bigg(\int_{\mathcal{T}}\rho\nabla\cdot\mathbf{u}_h^{n+1}\psi_j\,dx\Bigg)=0.\end{aligned}\tag{11}$$

Solving this linear system for $(\mathbf{u}_h^{n+1}, p_h^{n+1})$ constitutes one pseudo-time step. We write eq. (10) in matrix format

$$\frac{1}{\Delta t^n}M(\mathbf{v}^{n+1}-\mathbf{v}^n)+F(\mathbf{v}^{n+1})=0,\tag{12}$$

where the coefficients of $(\mathbf{u}_h^n, p_h^n)$ w.r.t. the chosen basis of X_h are again combined into a single vector $\mathbf{v}^n$, and where $M=\begin{bmatrix}M_{\mathbf{u}} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}$ is a block matrix with $M_{\mathbf{u}}$ the velocity mass matrix. Then, we can also write the solution of eq. (11) in the more compact form

$$\mathbf{v}^{n+1}=\mathbf{v}^n-(\frac{1}{\Delta t^n}M+F'(\mathbf{v}^n))^{-1}F(\mathbf{v}^n).\tag{13}$$

3.2. Choice of the pseudo-time step

The idea is that the pseudo-time step Δt^n is initially small, such that its influence in eq. (13) is large. When the iteration approaches convergence, the magnitude of the pseudo-time steps becomes large such that eq. (13) performs more like the standard Newton iteration. This has the benefit that, in the beginning, the method depends less on the initial guess, and in the end, it may behave closely to the quadratic convergence of Newton's method. There are several possibilities to define a pseudo-time step that fits this property.

In addition to that, it is possible to use a pseudo-time step that uses local, i.e. mesh element dependent, information. This amounts to replacing the global pseudo-time step Δt^n in eq. (11) by a local, mesh element dependent, pseudo-time step $\Delta t_{\mathcal{T}}^n$. In this case the mass matrix M in eq. (13) will depend on the vector of local pseudo-time steps $\overline{\Delta t}^n=(\Delta t_{\mathcal{T}_1}^n,\dots,\Delta t_{\mathcal{T}_m}^n)$, so that eq. (13) becomes

$$\mathbf{v}^{n+1}=\mathbf{v}^n-(M(\overline{\Delta t}^n)+F'(\mathbf{v}^n))^{-1}F(\mathbf{v}^n).\tag{14}$$

In COMSOL, for instance, the following local pseudo-time step is used:

$$\Delta t_{\mathcal{T}}^n=\mathrm{CFL}(n)\frac{h_{\mathcal{T}}}{\|\mathbf{u}_h^n\|_{\mathcal{T}}},\tag{15}$$

with $h_{\mathcal{T}}$ the size of mesh element $\mathcal{T}$, $\|\mathbf{u}_h^n\|_{\mathcal{T}}$ the Euclidean norm of the velocity field $\mathbf{u}_h^n$ in the mesh element $\mathcal{T}$, and $\mathrm{CFL}(n)$ a global Courant–Friedrichs–Lewy (CFL) number [57]. This global CFL number may depend on the iteration count n, for instance,

$$\mathrm{CFL}_{iter}(n)=\begin{cases}1.3^{\min(n,9)}, & 1\le n\le 20,\\ 1.3^9+9\cdot 1.3^{\min(n-20,9)}, & 20<n\le 40,\\ 1.3^9+9\cdot 1.3^9+90\cdot 1.3^{\min(n-40,9)}, & n>40;\end{cases}\tag{16}$$

see [57, p. 1108]. Alternatively, it can be given by a controller based on the nonlinear error estimate e_n for iteration n, the given target error estimate "tol", and control parameters k_P, k_I, and k_D, which are positive constants:

$$\mathrm{CFL}_e(n)=\left(\frac{e_{n-2}}{e_{n-1}}\right)^{k_P}\left(\frac{\mathrm{tol}}{e_{n-1}}\right)^{k_I}\left(\frac{e_{n-2}/e_{n-1}}{e_{n-3}/e_{n-2}}\right)^{k_D}\mathrm{CFL}_e(n-1),\tag{17}$$

where e_n is an estimate for the error in the nth nonlinear iteration; see [57, p. 1515]. Both these options are available for controlling the global CFL number in COMSOL.

For both cases, in the initial iterations of pseudo-time stepping, the method starts with a small global CFL number and gradually increases it as the solution reaches convergence [57]. In eq. (16) this is because the CFL number increases each iteration, also shown in Fig. 1. For eq. (17), when the solution is near convergence, the error estimates e_n will be smaller and thus the CFL number increases.

Note that, since we are not interested in following accurately the time evolution of the dynamical system eq. (4), the global CFL numbers in eqs. (16) and (17) are not chosen to satisfy a CFL condition for convergence of a time stepping scheme to the time-dependent solution; they are chosen to reach convergence to a stationary solution as robustly and quickly as possible. Other pseudo-time step control mechanisms include the "switched evolution relaxation" (SER) method [8].

In this paper we investigate replacing the pseudo-time step control algorithm by the use of an NN to control the local, mesh element dependent, pseudo-time step Δt_e^n. In order to make the method easily generalizable to different meshes, the NN is provided

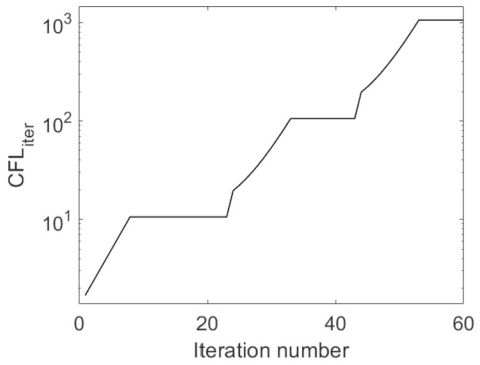


Fig. 1. A plot of CFL_{iter} depending on the nonlinear iteration count, as defined in eq. (16).

with local information from the corresponding mesh element and its adjacent mesh elements. One could think of local information about the solution, residuals, mesh and the cell Reynolds number. For example, elements with high local residuals could then have smaller local pseudo-time steps compared to elements with low residuals, which could accelerate convergence in areas which already have low residuals. So, treating each mesh element individually using local information can accelerate convergence for the whole simulation. In order to use these predictions in COMSOL, a continuous pseudo-time step function has to be defined withing COMSOL. This is done via interpolation: The local pseudo-time step size is predicted at the center of each mesh element. Then, the values are interpolated linearly between the centers of the mesh elements and extrapolated as a constant from the centers of the boundary elements towards the boundary. By doing so, the pseudo-time step is defined as a continuous function within the whole computational domain.

4. Neural network for the local pseudo-time step prediction

In order to predict an element-wise local pseudo-time step for the pseudo-time stepping that yields fast and robust Newton convergence, we employ a data-based approach based on machine learning techniques; more specifically, we use artificial neural networks (ANNs). The most basic form of an ANN is a multilayer perceptron (MLP) and is given as a composed function

$$\mathcal{NN}(x) = W_{n+1}\ f_n \circ \cdots \circ f_1(x),$$

with

$$f_i(x) = \sigma\left(W_i x + b_i\right). \tag{18}$$

Here, the $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and $b_i \in \mathbb{R}^{n_i}$ are the so-called weight matrices and bias vectors, which represent the linear parts of the NN function $\mathcal{NN}$; see Fig. 6 for an exemplary network architecture of an MLP. Given a specific network architecture, that is, the dimensions $\left(n_i\right)_i$, the MLP is determined by the coefficients of the weight matrices and bias vectors, also denoted as the network parameters. The MLP becomes nonlinear due to composition with the activation function σ. Here, we will employ the rectified linear unit (ReLU) function $\sigma(x) = \max(0, x)$. MLPs, and NNs in general, are well-suited for approximating nonlinear functions; see, e.g., [60]. In order fit an NN to input data $X = \{x_i\}$ and corresponding output data $Y = \{y_i\}$, a loss function $\mathcal{L}$ is minimized with respect to the network parameters:

$$\arg\min_{W_i, b_i} \mathcal{L}\left(\mathcal{NN}(X), Y\right)$$

The loss function penalizes deviation of the network function from the reference output data Y. In order to optimize the NN parameters (a.k.a. network training), we employ a stochastic gradient descent (SGD) method using adaptive moment estimation (Adam) [61], with an initial learning rate and decay of 0.001 every iteration; the gradients are computed using the backpropagation algorithm [62]. For a more detailed introduction to deep learning and NNs; see, e.g., [63].

In this section, we will discuss how the local pseudo-time steps used as reference data in the root mean squared error (RMSE) loss function are computed as well as the composition of our training data set and the network architecture.

4.1. Loss and optimal local pseudo-time step

Our goal is to predict local pseudo-time steps that enhance the pseudo-time stepping method in accelerating convergence of a given flow problem. In particular, we try to predict the vector of local pseudo-time steps $\overline{\Delta t}_{opt}$ such that the next iterate is as close as possible to the solution of the nonlinear problem. We will denote this as the *optimal local pseudo-time step* for an iterate $\mathbf{v}^n$, $\overline{\Delta t}_{opt}(\mathbf{v}^n)$, defined by

$$\overline{\Delta t}_{opt}(\mathbf{v}^n) := \arg\min_{\overline{\Delta t}} ||\mathbf{v}^n - \left(M(\overline{\Delta t}) + F'(\mathbf{v}^n)\right)^{-1} F(\mathbf{v}^n) - \mathbf{v}^*||, \tag{19}$$

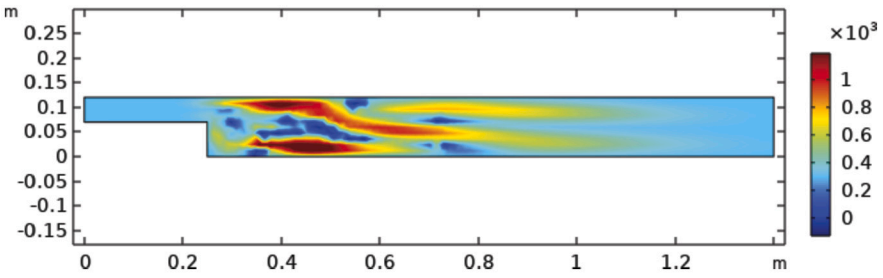


Fig. 2. The local pseudo-time step optimized by SNOPT; the difference between the obtained solution and the converged solution was minimized up to a value of $3.1 \cdot 10^{-5}$.

where $\mathbf{v}^*$ satisfies $F(\mathbf{v}^*) = 0$. The norm that is used here is the L^2 norm of the reconstructed velocity components.

Since the optimal local pseudo-time step cannot be computed explicitly, we compute an approximation via an optimization procedure, using the sparse nonlinear optimizer (SNOPT) software package [64,65]. This optimization is performed within COMSOL; see [54] for more details. For the example of a back-step geometry with laminar flow, we obtain the optimal local pseudo-time step as shown in Fig. 2 by using this two step procedure. We did not perform a detailed investigation of the computing time of the optimization, but the optimization for a single pseudo-time step took in the order of a few hours on employed hardware; cf. section 5. So the computational cost is considerable, but the optimization has to be performed only once for the data generation.

Once we have computed the local optimal pseudo-time step for all elements in the data set, we optimize the squared error of the network prediction against these reference values. The whole data set consists of a large number of optimal local pseudo-time steps $\Delta t_{e_i,opt}(\mathbf{v}^n)$ computed from various problem configurations, nonlinear iterates $\mathbf{v}^n$, and elements e_i computed using the optimization in COMSOL. To define the complete loss, let $\Delta t^{(i)}_{pred}$ and $\Delta t^{(i)}_{opt}$ be the network prediction and optimized pseudo-time steps, respectively, for the ith data point in a data set with n data points. The full RMSE loss, is then given as:

$$\text{loss} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\Delta t^{(i)}_{pred} - \Delta t^{(i)}_{opt}\right)^2}. \tag{20}$$

As mentioned earlier, we are interested in improving the nonlinear convergence rather than the temporal evolution. Therefore, negative pseudo-time steps are also valid, and we do not enforce any positivity constraint. Similarly, as discussed in [8], there is generally no upper limit for the pseudo-time step.

4.2. Neural network input

NNs are universal function approximators and can theoretically approximate any continuous nonlinear function up to arbitrary precision; see, for instance, [60,66,67]. Therefore, we expect that it is possible to predict the optimal local pseudo-time step, given sufficient input data. On the other hand, we want to limit the complexity of the input data and train a model with strong generalization properties. Therefore, similar to other approaches for NN-enhanced simulations [29,30,32], we employ local input data to predict the local optimal pseudo-time step. This means that we do not include any information about the specific global boundary value problem but only data that is given on a local patch of elements around the element of interest, that is, where the local optimal pseudo-time step is to be predicted. As a result, the trained NN model is applicable to any considered laminar flow problem. An additional benefit of using only local data is that the dimensionality and complexity of the data is being limited. Note that, as also noted in [29], due to the local data approach, a single simulation already generates a rich data set: each simulation yields data on a large set of elements over a number of Newton iterations. However, to generate a training data set with sufficient variation to allow for generalization of the model, the training data may be sampled from different simulations with varying flow conditions, induced by different boundary conditions and geometries as well as mesh refinement levels.

Let us now discuss the specific data included in the model input. As mentioned before, we employ data from a patch of elements

$$P_j = \bigcup_{i=1}^{4} e_{j,i}.$$

In particular, the patch consists of the element for which we want to compute the local optimal pseudo-time step, $e_{j,1}$, as well as the (up to) three adjacent elements $e_{j,2}$, $e_{j,3}$, and $e_{j,4}$; cf. Fig. 3. Specifically, we denote two elements as adjacent if they are connected via an element edge. On each patch, we sample the data listed in Table 1 with their respective location. The nodal information is explicitly available from the mixed finite element discretization used in our simulations; cf. the description of our simulation setup in section 2. At high CFL numbers, the ratio of propagation distance to element size increases. As a result, using input data from larger patches extending beyond a single element and its three adjacent elements could be beneficial in such cases. Although this aspect is beyond the scope of the current paper, it will be investigated in future work.

Let us briefly motivate our choices. Firstly, we include the element size, which is implied by the element edge lengths, and $\mathbf{u}$ since they are also employed in the computation of the pseudo-time step approach in eq. (15). We complement this $\mathbf{u}$ by p to provide a

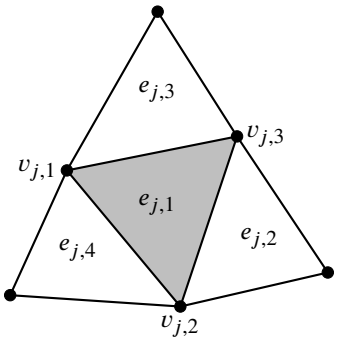


Fig. 3. Data point which contains information of a patch of four elements $e_{j,1}$ to $e_{j,4}$ with their vertices $v_{j,1}$ to $v_{j,6}$; see Table 1 for all input features for the NN model on each element patch.

Table 1
Variables used as input for the NN with their given unit and location on one element in the patch shown in Fig. 3, in this case the central element. One data point contains information of four elements, with a non-structured order of the adjacent elements. The two different types of residuals r_α and R_α, $\alpha = u, v, p$ are given in eqs. (21) and (22).

variable	unit	location
element edge length	m	-
u	m/s	$v_{j,1}$, $v_{j,2}$, $v_{j,3}$
v	m/s	$v_{j,1}$, $v_{j,2}$, $v_{j,3}$
p	Pa	$v_{j,1}$, $v_{j,2}$, $v_{j,3}$
R_u		$v_{j,1}$, $v_{j,2}$, $v_{j,3}$
R_v		$v_{j,1}$, $v_{j,2}$, $v_{j,3}$
R_p		$v_{j,1}$, $v_{j,2}$, $v_{j,3}$
r_u	N/m^3	$v_{j,1}$, $v_{j,2}$, $v_{j,3}$
r_v	N/m^3	$v_{j,1}$, $v_{j,2}$, $v_{j,3}$
r_p	kg/(m^3·s)	$v_{j,1}$, $v_{j,2}$, $v_{j,3}$
cell Reynolds number		$e_{j,1}$

complete description of the finite element solution on the local patch. Moreover, the local information describing the convergence of the Newton iteration is supplied in the form of two different types of residuals, i.e., the residual of the discretized system of nonlinear equations

$$F(\mathbf{v}) = \begin{pmatrix} R_u \\ R_v \\ R_p \end{pmatrix} = \begin{pmatrix} N(u,v) + B^\top p \\ B(u,v) \end{pmatrix} = \begin{pmatrix} N(\mathbf{u}) + B^\top p \\ B(\mathbf{u}) \end{pmatrix} \tag{21}$$

and the residual obtained by substituting the approximate solution into the system of PDEs eq. (1):

$$\begin{aligned} r_u &= \rho(u\partial_x u + v\partial_y u) - \mu(\partial_x^2 u + \partial_y^2 u) + \partial_x p + f_x, \\ r_v &= \rho(u\partial_x v + v\partial_y v) - \mu(\partial_x^2 v + \partial_y^2 v) + \partial_y p + f_y, \\ r_p &= \rho(\partial_x u + \partial_y v). \end{aligned} \tag{22}$$

The local Reynolds number, as given in eq. (2), provides additional information about the local nonlinearity of the flow; it is also used in papers [29,30,53] as input for the machine learning model. Furthermore, as shown in [54], using information from a patch of elements improves the convergence of the method more compared to using only information from a single element. The input is normalized vector-wise to have mean 0 and standard deviation by computing the z-score defined as

$$z(\sigma) = \frac{x - \mu}{\sigma},$$

with x the evaluated input data, μ the mean of the input, and σ the standard deviation of the input [68]. The mean and standard deviation for centering and scaling the data are saved and later on used to normalize input data from other simulations.

Note that, for practical reasons, in our implementation in COMSOL, we collect all input information, as listed in Table 1, element by element from each patch. This means that vertex- and edge-based information is duplicated where the elements in the patch touch. As a result, we obtain a total of $n_0 = 124$ input features for our network model. In practice, one may want to omit the redundant

Table 2
Dimensions of the back-step geometries considered for the training and test data; see Fig. 4 for the base geometry.

back-step geometry	dim. inflow tunnel (m)	dim. outflow tunnel (m)	range inflow velocity ($^m/_s$)	range maximum element size (m)
B1	0.05×0.25	0.12×1.15	0.001 – 0.015	0.0106 – 0.0256
B1S	0.005×0.025	0.012×0.115	0.01 – 0.15	0.00106 – 0.00256
B2	0.08×0.25	0.22×1.15	0.001 – 0.013	0.0126 – 0.0206
B2S	0.008×0.025	0.022×0.115	0.01 – 0.13	0.00126 – 0.00206

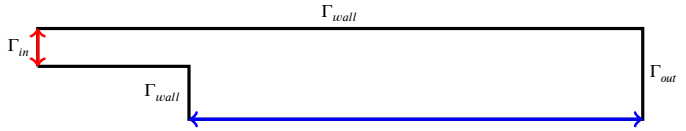


Fig. 4. Back-step geometries employed: B1, B1S, B2, and B2S vary in the width of the inlet Γ_{in} (red), where a constant velocity profile is prescribed, and the length of the blue wall segment; cf. Table 2. At the outlet Γ_{out}, we prescribe a constant pressure boundary condition, and at all other parts of the boundary, which are denoted as Γ_{wall}, a no-slip boundary condition is enforced. For an exemplary flow field, see Fig. 8.

data, however, this is currently not straightforward based on the data structures in our interface from Matlab to COMSOL since the elements in each patch cannot be easily retrieved in a consistent ordering. The neural network might be able to extract connectivity information from the redundant data, which could alternatively be encoded by a consistent ordering of the elements within the patch. Our numerical results in section 5 indicate that this handling of the input data does not prevent our model to learn from the data. However, in future work, we plan to investigate if using unique input data with a consistent ordering may have a positive effect on our model.

Finally, elements which are adjacent to the boundary of the computational domain Ω, may only have one or two neighboring elements, resulting in patches consisting of two or three elements only. In order to encode this in the data, we set any data of the vertices and edges outside the domain to zero; other approaches for encoding this information are possible and will, again, be considered in future work.

4.3. Generation of training data

We train our model based on simulation data from several boundary value problems defined on back-step geometries; see Fig. 2 for an exemplary back-step geometry. The boundary conditions for the back-step flow are defined as follows: We impose a constant inflow velocity profile at the inlet Γ_{in} and a constant pressure at the outlet Γ_{out} on the right side of the geometry. For the other parts of the boundary, Γ_{wall}, a no-slip boundary condition is prescribed.

As discussed in section 4.2, due to the locality of the input features, we expect that it suffices to use training data for a single type of geometry, as long as there is enough variation in the data with respect to the individual patches. In particular, we will consider two different base cases of back-step geometries, which we denote as B1 and B2. For both cases, we consider meshes with different levels of refinement and inflow velocities. This also results in flow fields with different Reynolds numbers. Moreover, we use data from different Newton iterations since, as can be seen in Fig. 8, the iterate changes quite drastically between different Newton iterations. Additionally, we consider scaled variants of B1 and B2, which we denote as B1S and B2S, respectively. These are obtained by scaling the geometric dimensions of B1 and B2 by a factor of 0.1 and increasing the inflow velocity by a factor of 10; cf. Table 2 for the dimensions of the four different geometries. As a result the Reynolds number remains the same, and we would expect the nonlinear convergence to be similar. By adding the scaled versions to the training configurations, we try to force the model to learn the dependence of the nonlinearity on the Reynolds number. In the case of the back-step, the step height is considered as the characteristic length to compute the Reynolds number. The considered dynamic viscosity μ is $9.98 \cdot 10^{-4}\ mPa \cdot s$ and the density ρ is $988\ ^{kg}/_{m^3}$. We summarize all configurations considered for the generation of training data in Table 3. All these configurations are in the laminar regime, allowing for a stable stationary solution, which we checked by running a time dependent solver.

In total, we obtain 79 256 data points from all configurations listed in Table 3. To arrive at a balanced distribution of the training data set, we perform the sampling such that we obtain roughly the same number of data points for each element size in the data set; that is, we select exactly 3 500 data points from all configurations with the same maximum element size. In total, we can categorize 14 different simulations based on the maximum element size, which gives a data set of 49 000 data points. To make it easier to apply batch learning for the NN, 1 000 data points are discarded at random. So, in total there are 48 000 data points, of which 70% are used for training our network model, 15% are used for testing and 15% are used for validation. In section 5, we will first discuss how this model performs on configurations with back-step geometry in section 5.1. Then, in sections 5.2 and 5.3, we will also discuss the generalization to other configurations, that is, to Couette flow and flow around an obstacle, respectively.

Table 3
Different combinations of mesh sizes and flow velocities of the back-step simulations to generate training data. The number of obtained data points is given in column 2, which are sampled from the nonlinear iteration steps given in the last column. The dimensions of the four back-steps can be found in Table 2. Here, # iterations denote the number of nonlinear iterations performed before extracting the data.

back-step geometry	number of elements	max. element size (m)	inflow velocity ($^m/_s$)	# iterations	Re
B1	6 516	0.0156	0.001	1, 10	78
			0.005	10	392
			0.01	10	784
	3 908	0.0206	0.003	1, 10	235
			0.006	10	470
			0.009	10	706
	3 558	0.0266	0.002	1, 10	157
			0.007	10	549
			0.008	2, 4	627
			0.015	10	1 176
	13 828	0.0106	0.004	2, 10	313
			0.008	2, 10	627
B1S (scaled B1)	6 516	0.00156	0.01	1, 10	78
			0.05	10	392
			0.1	10	784
	3 908	0.00206	0.03	1, 10	235
			0.06	10	470
			0.09	10	706
	3 558	0.00266	0.02	1, 10	157
			0.07	10	549
			0.08	2, 4	627
			0.15	10	1 176
	13 828	0.00106	0.04	2, 10	313
			0.08	2, 10	627
B2	5 808	0.0156	0.005	1, 10	784
	4 134	0.0186	0.01	2, 10	1 568
	8 790	0.0126	0.013	3, 10	2 039
B2S (scaled B2)	5 808	0.00156	0.05	1, 10	784
	4 134	0.00186	0.1	2, 10	1 568
	8 790	0.00126	0.13	3, 10	2 039

Table 4
Hyper parameters and search space for the grid search.

hyper parameter	search space
# hidden layers	$\{2, 3, 4, 5\}$
# neurons per hidden layer	$\{2^4, 2^5, 2^6, 2^7, 2^8\}$

4.4. Network architecture and training

The architecture of the network model employed in this work is depicted in Fig. 6: the model consists of an input layer with 124 neurons, two hidden layers with 16 neurons each, and an output layer with the neuron representing prediction of the local optimal pseudo-time step; the composition of the input data has been discussed in section 4.2.

We determined the network architecture as follows: We first performed a grid search with 6-fold cross validation on the search space given in Table 4 to determine three different network architectures with an acceptable average validation loss. Note that the loss is defined based on the prediction of the local optimal pseudo-time step; that means a low validation loss does not necessarily result in a low solution residual when the network is used. From the grid search, we determined one network with small, one with medium, and one with high networks capacity, that is, networks with:

- 2 hidden layers and 16 neurons per layer,
- 3 hidden layers and 64 neurons per layer,
- 4 hidden layers and 256 neurons per layer.

Then, in order to find the best model for accelerating the Newton convergence, we compare them when applied to several back-step and Couette simulations, further explained in section 5.1 and section 5.2. The combinations of the inflow or wall velocities with the maximum element size used for these simulations are given in Table 5.

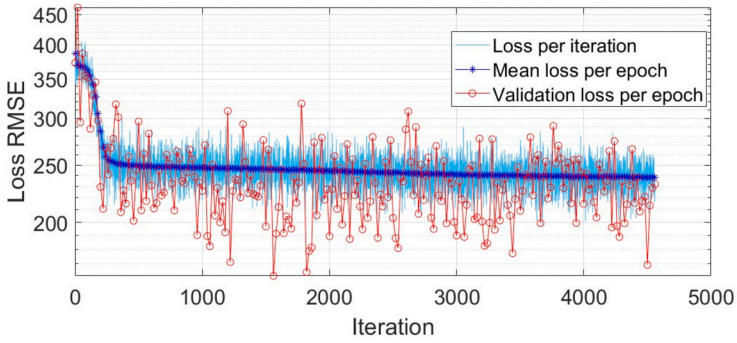


Fig. 5. Training plot of network trained on the optimized pseudo-time step targets. Every epoch consists of 20 iterations, each iteration with a mini-batch size 1225. With a validation patience of 150 epochs, the smallest validation loss for this network is obtained after 78 epochs. The training costed in total 228 epochs in a time of 9:31 minutes.

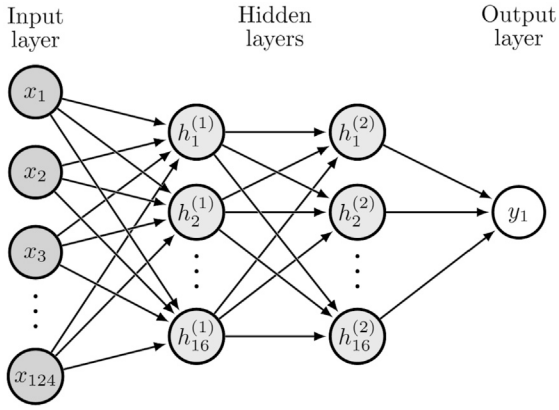


Fig. 6. Exemplary NN with 124 inputs, 2 hidden layers with 16 neurons and an output layer. The arrows are associated with weights in the weight matrices W_i; cf. eq. (18).

Table 5
Combinations of velocities and mesh sizes for simulations used to test the NN performance, with the back-step configurations B1–B2S in Table 2 and the Couette flow configurations C and CS.

geometry type	inflow/wall velocity ($^m/_s$)	max. element size (m)	Reynolds number
B1	0.001, 0.004, 0.007, 0.01, 0.012, 0.015	0.0106 : 0.005 : 0.0256	76 – 1153
B1S	0.01, 0.04, 0.07, 0.1, 0.12, 0.15	0.00106 : 0.0005 : 0.00256	76 – 1153
B2	0.001 : 0.003 : 0.01	0.0126, 0.0156, 0.0186, 0.0206	153 – 1537
B2S	0.01 : 0.03 : 0.1	0.00126, 0.00156, 0.00186, 0.00206	153 – 1537
C	0.01, 0.03, 0.04, 0.05, 0.07, 0.1	0.014 : 0.002 : 0.022	2195 – 21 955
CS	0.01, 0.03, 0.05	0.0028 : 0.0004 : 0.0044	439 – 2195

For each network, the required number of nonlinear iterations per simulation was compared. Surprisingly, the largest network now performed much worse, while the smallest network performed best, which could be explained by an overfitting behavior.

Each network is trained based on the different back-step simulations given in Table 2. As mentioned before, the local optimal pseudo-time steps obtained using the SNOPT optimizer are used as the reference. In particular, the network output is compared to these optimized local pseudo-time steps via the RMSE loss function eq. (20). As the optimizer, we use stochastic gradient descent with adaptive moments (Adam) [61] with an initial learning rate of 0.001. As the stopping criterion and regularization, we employ early stopping with a patience of 150 epochs. The evolution of the loss during training of the NN is shown in Fig. 5.

5. Numerical results

In this section, we discuss numerical results for our NN-enhanced pseudo-time stepping approach, which we have introduced in the previous section 4. As discussed in section 4.3, we have trained the model only for back-step geometries. Our model is designed such that only local features are used as input in order to enable generalizability. First, we test our model on the *back-step flow* training cases, with varying mesh sizes and inflow velocities within the range of the training data; cf. section 5.1. In addition to that, we investigate the performance on flipped and rotated cases. Then, in order to further test the generalization properties of the model,

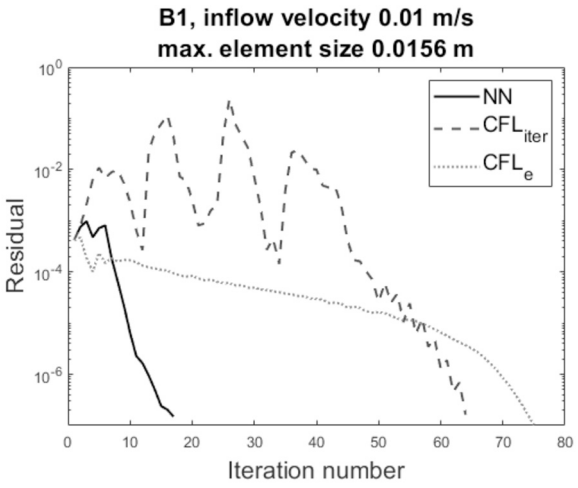


Fig. 7. Convergence plot of a B1 simulation for all three strategies for the local pseudo-time step under consideration: using the NN model as well as the strategies defined in eqs. (16) and (17).

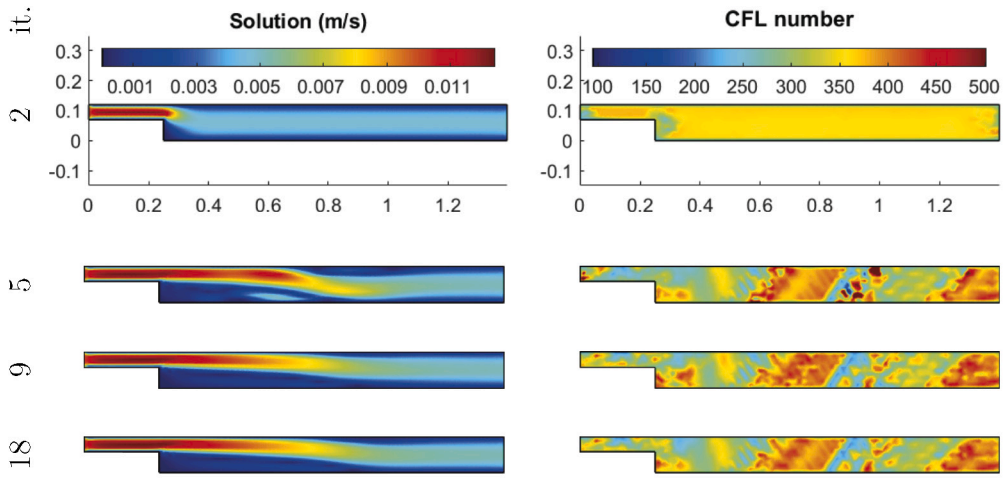


Fig. 8. Iterations of B1 with inflow velocity 0.001 m/s and maximum element size 0.0156 m using NN, with velocities (left) and corresponding distribution of the predicted local pseudo-time step prediction (right).

we test our model on *Couette cylinder flows* in section 5.2 as well as for *flow around an obstacle* cases in section 5.3. For the Couette cases, we vary the geometry, the mesh size, and the velocity of the inner rotating wall, and for the flow around an obstacle, back-step and Couette cylinder geometries with different kinds of obstacles are considered.

In particular, we compare the convergence of our hybrid approach with the convergence of two classical strategies for the choice of the local pseudo-time step described in section 3.2, that is, the strategy denoted by CFL_{iter}, as given by eqs. (15) and (16), and the strategy denoted by CFL_e, as given by eqs. (15) and (17). Furthermore, in the appendix, the convergence of the network is also compared with two variants of Newton's method, that is, the standard Newton method with a constant damping factor and a variant with an adaptive choice of the damping factor, as well as Newton's method with Anderson acceleration; see appendices A and B for more details. The initial condition of all the simulations is the zero velocity field. The L^2 norm of the "reconstructed" residual $\tilde{R}$

$$\|\tilde{R}\|_{L^2(\Omega)} = \sqrt{\int_\Omega \tilde{R}_u^2 + \tilde{R}_v^2 + \tilde{R}_p^2 \, dx} \tag{23}$$

is used as the stopping criterion. Here, $(\tilde{R}_u, \tilde{R}_v)$ and $\tilde{R}_p$ are the residual velocity and pressure fields in X_h reconstructed by using the components of the residual vector $R = (R_u, R_v, R_p)$, as defined in eq. (21), as coefficients w.r.t. the chosen basis of X_h.

Note that COMSOL uses a scaled version of the residual eq. (23) in the stopping criterion, such that the tolerance may differ depending on the problem configuration. However, when reporting the convergence results, we make sure that the same tolerance for the unscaled residual eq. (23) is used to determine convergence for the different approaches.

All simulations have been performed using COMSOL Multiphysics® version 6.1 on an Intel Core i7-6820HQ CPU, the NN computations have been carried out on an NVIDIA Quadro M100M GPU using Matlab.

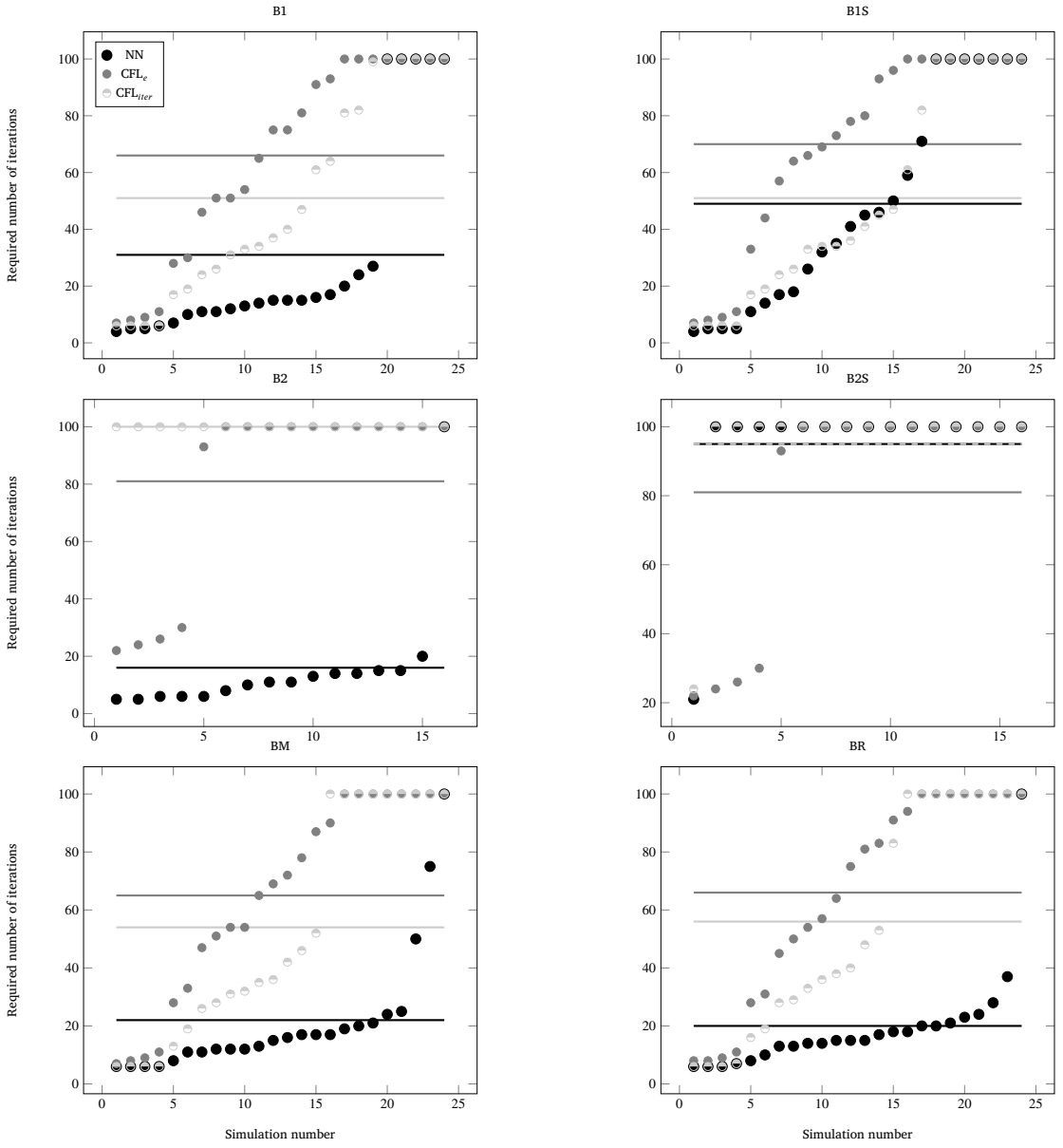


Fig. 9. Back-steps results for each method; the simulations are ordered from smallest required number of nonlinear iterations to largest separately for each approach. The horizontal lines indicate the average nonlinear iteration counts for the different approaches.

5.1. Back-step geometries

In this subsection, we discuss the performance of our model on the same four back-step geometries which we also used for the training; cf. section 4.3 and Table 2. In particular, we investigate if the network can accelerate the convergence when varying inflow velocity and mesh size on those configurations; cf. Table 5. As a result of those variations, a total of 80 different simulations for the back-step geometries are analyzed.

Exemplary, we plot the convergence history for the back-step geometry B1 with inflow velocity 0.001 $^{m}/_{s}$ and a maximum element size of 0.0156 m in Fig. 7. It can be observed that the network speeds up the convergence significantly compared with the two reference approaches; moreover, whereas the residual norm oscillates strongly for CFL$_{iter}$, the convergence is almost monotonous for the network and CFL$_e$ approaches. The qualitative behavior observed in Fig. 7, where the classical choices for the CFL number yield slow convergence or even strong oscillations in the residual, is not an exception; similar convergence behavior can also be observed for Couette flow and flow around an obstacle configuration; cf. sections 5.2 and 5.3. Furthermore, in Fig. 8, we depict four iterates during the Newton iteration using the network approach as well as the corresponding predicted local optimal pseudo-time step. It can

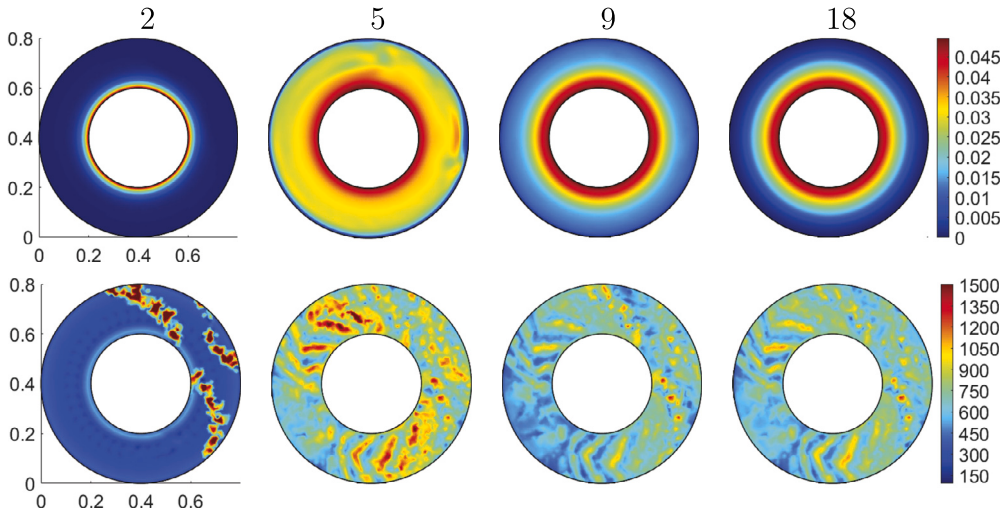


Fig. 10. Iterations of C with wall velocity 0.05 $^{m}/_{s}$ and maximum element size of 0.016 m using NN, with velocity (above) and local pseudo-time step prediction (below). The shown asymmetry in the pseudo-time step prediction is probably due to a not completely rotationally symmetric mesh.

be observed that both the iterate and predicted local optimal pseudo-time step still vary during the first iterations and both become stationary in the later iterations.

In Fig. 9, we compare the performance with respect to the numbers of Newton iterations resulting from the different choices NN, CFL_{iter}, and CFL_{e} for the pseudo-time step on different back-step configurations; cf. section 4.3 and Table 2. Furthermore, as a first investigation of the generalizability, we additionally consider cases resulting from mirroring (BM) or anti-clockwise rotating by 90 degrees (BR) the B1 cases.

For most cases from our training data B1–B2S, the NN model performs better than or equally well as the best choice of CFL_{iter} and CFL_{e}, as well as both Newton methods; see appendix A for the results for the Newton methods. This can also be seen in the average number of iterations for the configurations B1, B2, BM, and BR; see Figs. 9 and A.15. For B1S, the network performs equally well as CFL_{iter}.

In particular, for cases with lower velocities (B1 and B2), the network performs best. For cases with higher velocities (B1S and B2S) but the same Reynolds numbers, the network model performs clearly worse compared to the low velocity cases. Moreover, it could be observed that, for B1S and B2S, convergence was generally less robust; in particular, we observed a relatively high ratio of cases for which none of the pseudo-time step approaches converged. Since the nonlinearity should be mostly determined by the Reynolds number rather than the magnitude of the velocities, we would like to investigate in future research whether a scaling/normalization of the network inputs can improve the performance. The same behavior can also be observed for the Newton methods, although these methods converge in more cases; cf. appendix A.

Even though the mirrored and rotated configurations, of course, yield the same but mirrored respectively rotated results, we observe slight variations in the performance of the network model. This shows a slight overfitting with respect to the B1 configurations as shown exemplarily in Fig. 8, where, for instance, the velocities in positive x direction are dominant; this clearly changes when the geometry is mirrored or rotated. However, the performance of the network model is only slightly worse compared to the B1 cases, which shows that the overfitting is not severe with respect to the prediction of suitable local pseudo-time steps. In the future, we may consider group invariant networks [69] to prevent these geometric overfitting effects.

5.2. Couette

Next, we discuss the generalization of our hybrid globalization approach to a different type of geometries, that is, Couette cylinders. We consider two different geometries and a range of configurations resulting from varying the wall rotation velocity and refinement of the mesh; C with an outer radius of 0.4 m and an inner radius of 0.2 m, and CS with an outer radius of 0.08 m and an outer radius of 0.04 m. In total, we obtain 45 different configurations of Couette simulations, and in Table 5, the corresponding boundary conditions and maximum element sizes for the back-step configurations previously used for training the NN model can be found. Here, the characteristic length used to compute the Reynolds number is the distance between the two cylinders.

Again, we present the iterates and corresponding predicted local pseudo-time steps during the Newton iteration for one exemplary case with C geometry; see Fig. 10. As for the back-step configuration in Fig. 8, we observe that both the solution and local pseudo-time step prediction approach a stationary distribution.

Figs. 11 and A.16 confirm that the use of our NN-based approach, which has only been trained on back-step geometries, is beneficial for most of the Couette flow cases. First of all, we notice that, for the Couette flow simulations considered, all choices for the local pseudo-time step yield convergence. However, the NN-based approach accelerates convergence in 70% of the C cases and even 93.3% of the CS cases. The fact that the network seems to perform better for CS than for C cases, compared with the CFL_{iter}

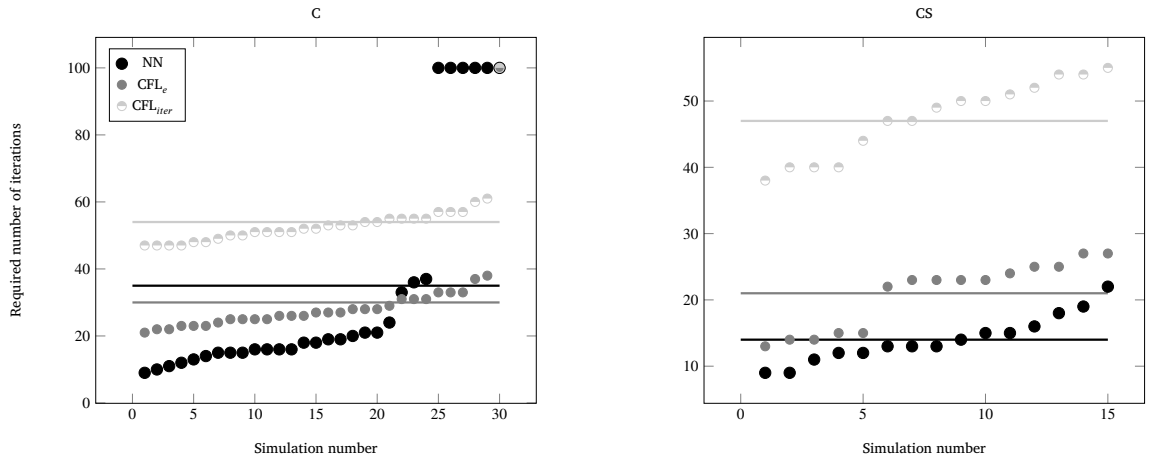


Fig. 11. The results of the Couette flow for each method; the simulations are ordered from smallest required number of nonlinear iterations to largest separately for each approach. The horizontal lines indicate the average nonlinear iteration counts for the different approaches.

Table 6
Combinations of object center coordinates, velocities and mesh. Here, BO are B1 back-steps in Table 2 with obstacles: a circle with radius 0.03 m, an ellipse with a-semiaxis 0.03 m and b-semiaxis 0.02 m, both with inflow velocity of 0.003 $^m/_s$ and an ellipse with a-semiaxis 0.02 m and b-semiaxis 0.03 m with inflow velocity of 0.002 $^m/_s$. The Couette flow has geometry of CS with outer radius 0.08 m and inner radius 0.04 m with obstacles: a circle with radius 0.008 m, an ellipse with a-semiaxis 0.01 m and b-semiaxis 0.006 m, and an ellipse with a-semiaxis 0.006 m and b-semiaxis 0.01 m.

geometry type	x (m)	y (m)	inflow/wall velocity ($^m/_s$)	max. element size (m)
BO	0.37	0.035 : 0.005 : 0.08	0.002, 0.003	0.0126
BO	0.3, 1.1	0.04	0.002, 0.003	0.0126
CO	0.004	0.016	0.006 : 0.002 : 0.01	0.0028 : 0.0004 : 0.0044

and CFL_e choices, is remarkable because the CS cases contain higher velocities and smaller mesh element sizes compared to C. For the back-step geometries, we observed the opposite; cf. Fig. 9, where the network performed worse on B1S and B2S than B1 and B2 configurations.

Also when taking the two variants of Newton's method into account in the results in appendix A, the NN approach remains competitive. In particular, the NN is on average close to the best method (for the C configurations) or as good as the best method (for the CS configurations).

5.3. Flow around an obstacle

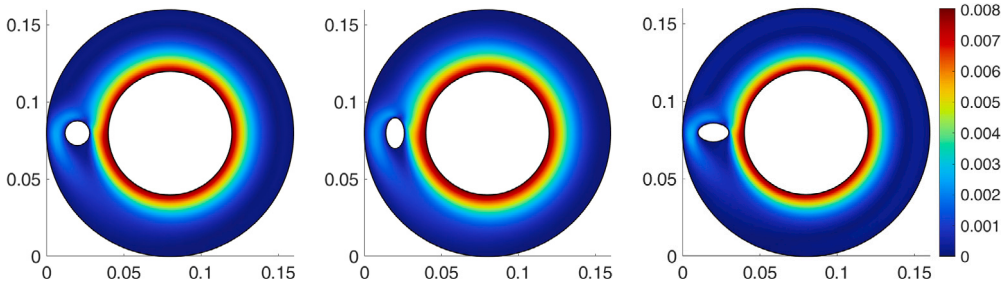
As the final type of test examples, we consider Couette and back-step flow around an obstacle. In particular, we place an obstacle inside the computational domain of the back-step B1 and Couette C configurations; cf. Table 6, where different obstacle geometries are considered: circles with different radii and ellipses with different a-semiaxis and b-semiaxis. We denote the resulting configurations as back-step with obstacle (BO) and Couette cylinder with obstacle (CO); see Table 6 for more details on the configurations and Fig. 12 for exemplary flow fields for the three different obstacle geometries in Couette flow. In total, we obtain 36 BO and 45 CO configurations. The Reynolds numbers of the back-step with obstacle lay between 60 and 90, and for the Couette flow with obstacle between 35 and 100, taking the obstacle hight as the characteristic length.

The results of the simulations with obstacles are given in Figs. 14 and A.16. Qualitatively, the NN approach compares to CFL_{iter} and CFL_e similarly as before. It performs better on 36.3% of the BO cases and on all of the CO cases. Note that for the BO cases, the NN approach performs around the same as the CFL_e approach. Also, the average number of iteration is best for the NN approach for the BO and CO configurations. For one of the considered simulations of CO with an elliptic obstacle, we again plot iterates and corresponding distributions of the local pseudo-time step in Fig. 13.

Finally, also compared with the Newton methods in Appendix A, the NN performs clearly best, that is, both in terms of robustness and in terms of acceleration of convergence.

6. Conclusion

We have introduced a novel NN-based approach to improve the pseudo-time stepping algorithm. Instead of using local pseudo-time steps based on predefined global CFL numbers, optimal local pseudo-time step predictions of the network are used to compute the local pseudo-time step. The network uses local information in order to make these local pseudo-time step predictions and to



(a) Circular obstacle with diameter of 0.08 m.

(b) Elliptic obstacle with a-semiaxis 0.006 m and b-semiaxis 0.01 m.

(c) Elliptic obstacle with a-semiaxis 0.01 m and b-semiaxis 0.006 m.

Fig. 12. Velocity solutions for Couette flow with different kinds of obstacles. The wall rotation is 0.008 $^{m}/_{s}$ and the maximum element size is 0.0032 m. The center of the obstacle is x = 0.02 m and y = 0.08 m.

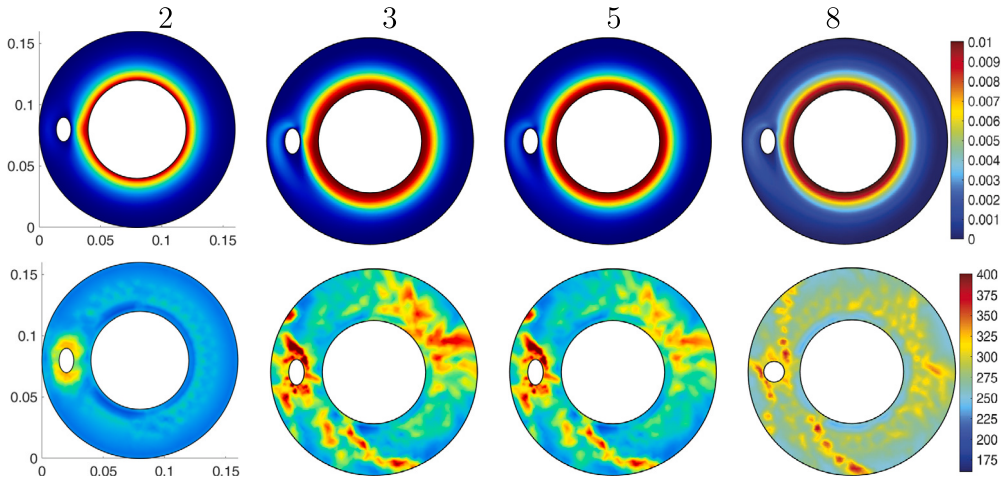


Fig. 13. Iterations of CO with wall velocity 0.01 $^{m}/_{s}$ and maximum element size of 0.004 m using NN, with velocity (above) and local pseudo-time step prediction (below). The obstacle is an ellipse with a-semiaxis 0.006 m and b-semiaxis 0.01 m, with center x = 0.02 m and y = 0.08 m.

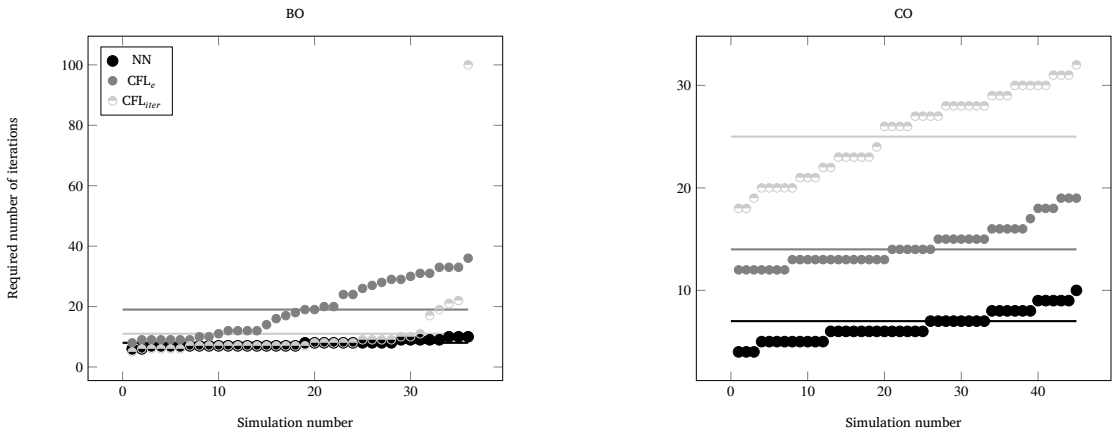


Fig. 14. The results of the simulations with obstacles for each method; the simulations are ordered from smallest required number of nonlinear iterations to largest separately for each approach. The horizontal lines indicate the average nonlinear iteration counts for the different approaches.

achieve generalizability. As a result, it can accelerate the convergence for the simulations on which it has been trained as well as for simulations which it has not seen before. In all considered simulations, the network was able to perform better or equally well compared to the standard CFL number based pseudo-time stepping strategies in most cases; the same is true in comparison with the variants of Newton's method.

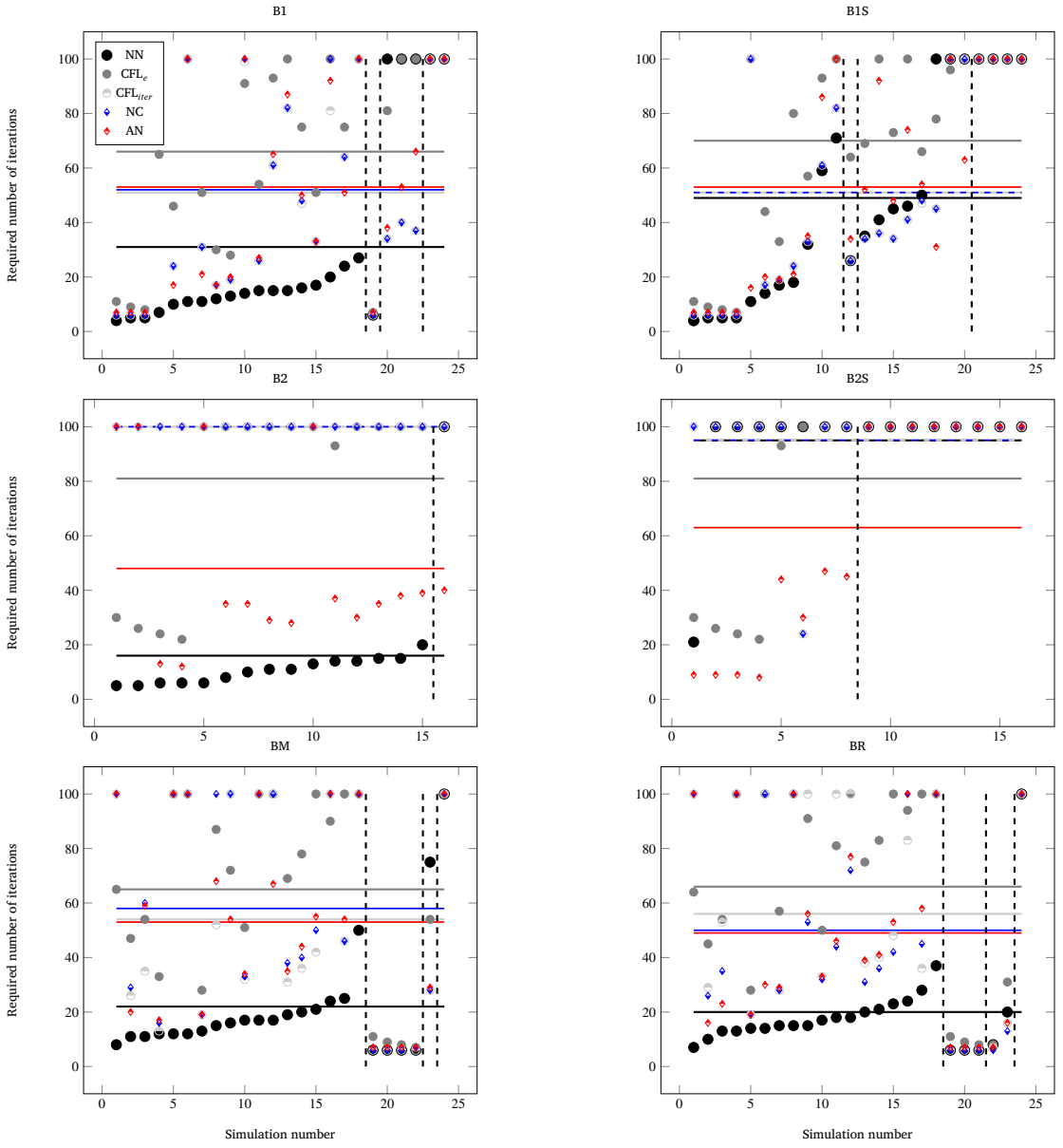


Fig. A.15. Back-step flow results using all approaches under consideration: using the NN, the strategies defined in eqs. (16) and (17), model as well as the NC and AN Newton methods; see appendix A for details on the organization of the plots.

In future work, we would like to extend the method to turbulent and three-dimensional flow cases. Furthermore, we will try to improve the method presented here, for example, by generating a more diverse training data set, combining the heuristic strategies for choosing the pseudo-time step with our network approach the default CFL number based strategies and the network, or removing redundancies in the input features. Furthermore, the impact of the patch size containing the neural network inputs should be examined, as larger patches may be particularly advantageous for high-speed fluid flows.

Appendix A. Comparison against variants of Newton's method

In Figs. A.15 and A.16, we show results for all configurations considered, comparing our novel NN approach with four nonlinear solvers implemented in COMSOL, that is, the two default CFL number strategies defined in eqs. (16) and (17) as well as two variants of Newton's method. As variants of Newton's method, we employ the standard Newton method with damping factor of one and an adaptive choice of the damping factor, depending on the intermediate iterates. We denote the two approaches as Newton constant (NC) and automatic Newton (AN), respectively; cf. [57, p. 1627]. Different from Figs. 9, 11, and 14, the ordering of the simulations for the different methods is the same for each approach. The simulations are first organized into different blocks, and within each

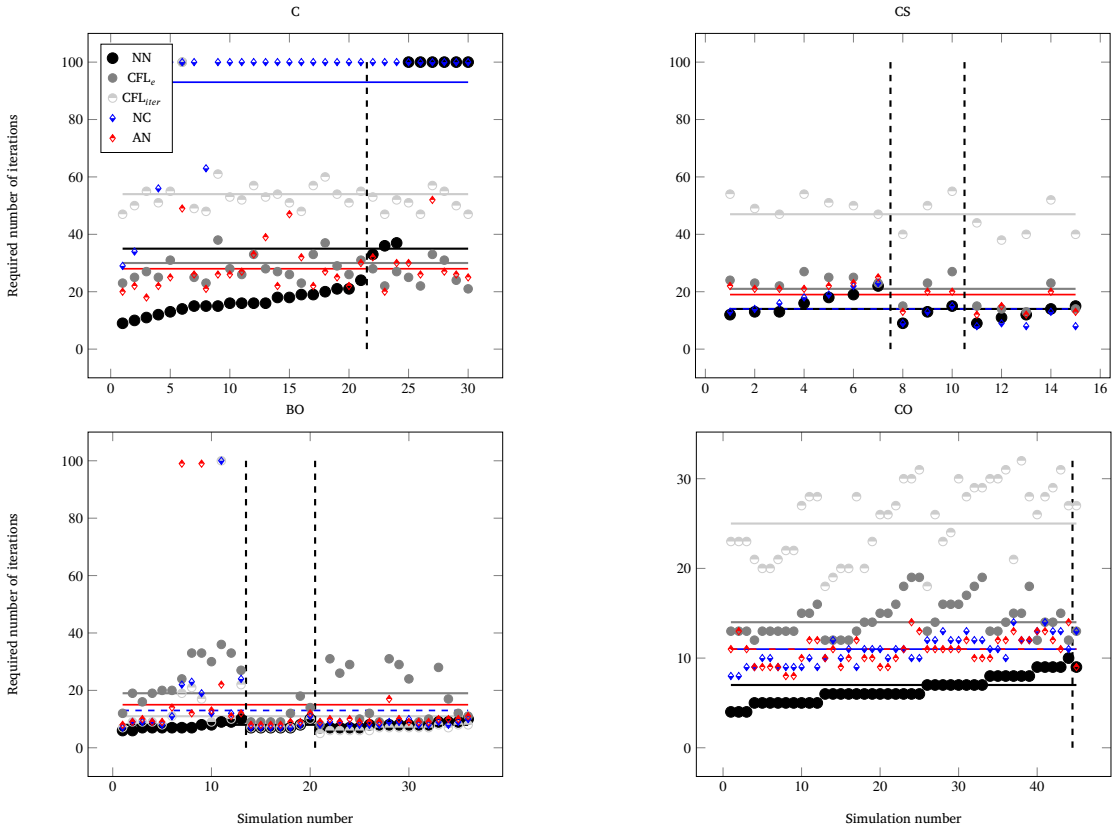


Fig. A.16. Couette flow and flow around an obstacle results using all approaches under consideration: using the NN, the strategies defined in eqs. (16) and (17), model as well as the NC and AN Newton methods; see Appendix A for details on the organization of the plots. The horizontal lines indicate the average nonlinear iteration counts for the different approaches.

block the ordering is based on the number of nonlinear iterations needed for the NN approach. The blocks are as follows: before the first vertical dashed line, the NN performed best; between the first and second vertical dashed line, the NN performed best with the same number of iterations as at least one other method; between the second and third vertical dashed line, the network performed worse than at least one other method; after the third vertical line, none of the methods converged.

The results are mostly in alignment with the previous results as the NC and AN methods often perform similarly to one of the classical CFL strategies considered. In particular, when inspecting the average iteration count, both NC and AN perform quite similarly to CFL$_{iter}$ for B1, B1S, BM, BR, CS, and BO configurations; in all those cases the NN approach is clearly competitive. For the CO and B2 cases, AN performs better than NC, while the NN approach performs best. Only for the B2S configuration, where the NN approach generally performed worst, AC seems to perform clearly best; however, all approaches struggle for the B2S cases. Finally, for the C configurations, all methods perform similarly well, except for NC, which fails in most of the cases.

In summary, adding the NC and AN approaches to the picture does not change the qualitative comparison much; only for the B2S cases, AN performs clearly better than the NN approach. In future work, this should be taken into account for further improving the NN model.

Appendix B. Comparison against Anderson acceleration

The performance of the network has also been compared to the performance of the NC approach applying Anderson acceleration [70]. Therefore, the default settings for the Anderson acceleration in COMSOL are used, that is the dimension of iteration space, mixing factor, and iteration delay are respectively set to 10, 1.0, and 0, respectively. Again, we observe that, on average, the NN approach performs better for the simulations in Tables 5 and 6, except for the BS2 configurations. In 66 percent of the simulations, the NN resulted in an improvement with respect to Anderson acceleration. Furthermore, the NN improved the amount of converged simulations by 16.

Data availability

The generated training data of the back-step simulations, as well as the data of each column of the simulations mentioned in Table 3 and data with all the numerical results are available at https://github.com/searhein/nn-pseudo-time-stepping-data.

References

[1] R.P. Pawlowski, J.N. Shadid, J.P. Simonis, H.F. Walker, Globalization techniques for Newton-Krylov methods and applications to the fully coupled solution of the Navier-Stokes equations, SIAM Rev. 48 (4) (2006) 700–721, https://doi.org/10.1137/S0036144504443511.

[2] J.E. Dennis, R.B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Classics in Applied Mathematics, Society for Industrial and Applied Mathematics, 1996.

[3] S.C. Eisenstat, H.F. Walker, Globally convergent inexact Newton methods, SIAM J. Optim. 4 (2) (1994) 393–422, https://doi.org/10.1137/0804022, publisher: Society for Industrial and Applied Mathematics.

[4] E.L. Allgower, K. Georg, Continuation and Path Following, Acta Numerica, vol. 2, Cambridge University Press, 1993, pp. 1–64.

[5] K.A. Cliffe, A. Spence, S.J. Tavener, The Numerical Analysis of Bifurcation Problems with Application to Fluid Mechanics, Acta Numerica, vol. 9, Cambridge University Press, 2000, pp. 39–131.

[6] P. Cresta, O. Allix, C. Rey, S. Guinard, Nonlinear localization strategies for domain decomposition methods: application to post-buckling analyses, Comput. Methods Appl. Mech. Eng. 196 (8) (2007) 1436–1446, https://doi.org/10.1016/j.cma.2006.03.013.

[7] F.-N. Hwang, X.-C. Cai, A parallel nonlinear additive Schwarz preconditioned inexact Newton algorithm for incompressible Navier–Stokes equations, J. Comput. Phys. 204 (2) (2005) 666–691, https://doi.org/10.1016/j.jcp.2004.10.025.

[8] C. Kelley, D. Keyes, Convergence analysis of pseudo-transient continuation, SIAM J. Numer. Anal. 35 (08 1996), https://doi.org/10.1137/S0036142996304796.

[9] D.A. Knoll, D.E. Keyes, Jacobian-free Newton–Krylov methods: a survey of approaches and applications, J. Comput. Phys. 193 (2) (2004) 357–397, https://doi.org/10.1016/j.jcp.2003.08.010.

[10] M. Kubíček, M. Marek, Computational Methods in Bifurcation Theory and Dissipative Structures, Springer, Berlin, Heidelberg, 1983.

[11] L. Liu, F.-N. Hwang, L. Luo, X.-C. Cai, D.E. Keyes, A nonlinear elimination preconditioned inexact Newton algorithm, SIAM J. Sci. Comput. 44 (3) (2022) A1579–A1605, https://doi.org/10.1137/21M1416138, publisher: Society for Industrial and Applied Mathematics.

[12] L.T. Watson, Globally convergent homotopy algorithms for nonlinear systems of equations, Nonlinear Dyn. 1 (2) (1990) 143–191, https://doi.org/10.1007/BF01857785.

[13] L.T. Watson, M. Sosonkina, R.C. Melville, A.P. Morgan, H.F. Walker, Algorithm 777: HOMPACK90: a suite of Fortran 90 codes for globally convergent homotopy algorithms, ACM Trans. Math. Softw. 23 (4) (1997) 514–549, https://doi.org/10.1145/279232.279235.

[14] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, K. Willcox, S. Lee, Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence, Tech. Rep., USDOE Office of Science (SC), Washington, D.C. (United States), Feb. 2019.

[15] S.L. Brunton, B.R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annu. Rev. Fluid Mech. 52 (1) (2020) 477–508, https://doi.org/10.1146/annurev-fluid-010719-060214.

[16] S. Cai, Z. Mao, Z. Wang, M. Yin, G.E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: a review, Acta Mech. Sin. 37 (12) (2021) 1727–1738, https://doi.org/10.1007/s10409-021-01148-1.

[17] I. Lagaris, A. Likas, D. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000, https://doi.org/10.1109/72.712178.

[18] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, https://doi.org/10.1016/j.jcp.2018.10.045.

[19] M. Eichinger, A. Heinlein, A. Klawonn, Surrogate convolutional neural network models for steady computational fluid dynamics simulations, Electron. Trans. Numer. Anal. 56 (2022) 235–255, https://doi.org/10.1553/etna_vol56s235.

[20] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 481–490.

[21] B. Kim, V.C. Azevedo, N. Thuerey, T. Kim, M. Gross, B. Solenthaler, Deep fluids: a generative network for parameterized fluid simulations, Comput. Graph. Forum 38 (2) (2019) 59–70, https://doi.org/10.1111/cgf.13619, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13619.

[22] M. Kemna, A. Heinlein, C. Vuik, Reduced order fluid modeling with generative adversarial networks, PAMM 23 (1) (2023) e202200241, https://doi.org/10.1002/pamm.202200241.

[23] A. Beck, D. Flad, C.-D. Munz, Deep neural networks for data-driven LES closure models, J. Comput. Phys. 398 (2019) 108910, https://doi.org/10.1016/j.jcp.2019.108910.

[24] M. Schmelzer, R.P. Dwight, P. Cinnella, Discovery of algebraic Reynolds-stress models using sparse symbolic regression, Flow Turbul. Combust. 104 (2) (2020) 579–603, https://doi.org/10.1007/s10494-019-00089-x.

[25] S.L. Brunton, J.L. Proctor, J.N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proc. Natl. Acad. Sci. USA 113 (15) (2016) 3932–3937, https://doi.org/10.1073/pnas.1517384113.

[26] S. Wiewel, M. Becher, N. Thuerey, Latent space physics: towards learning the temporal evolution of fluid flow, Comput. Graph. Forum 38 (2) (2019) 71–82, https://doi.org/10.1111/cgf.13620.

[27] K. Huang, M. Krügener, A. Brown, F. Menhorn, H.-J. Bungartz, D. Hartmann, Machine learning-based optimal mesh generation in computational fluid dynamics, arXiv:2102.12923 [physics], Feb. 2021, https://doi.org/10.48550/arXiv.2102.12923.

[28] D. Kochkov, J.A. Smith, A. Alieva, Q. Wang, M.P. Brenner, S. Hoyer, Machine learning–accelerated computational fluid dynamics, Proc. Natl. Acad. Sci. USA 118 (21) (2021) e2101784118, https://doi.org/10.1073/pnas.2101784118.

[29] N. Margenberg, D. Hartmann, C. Lessig, T. Richter, A neural network multigrid solver for the Navier-Stokes equations, J. Comput. Phys. (2021) 110983, https://doi.org/10.1016/j.jcp.2022.110983.

[30] N. Margenberg, R. Jendersie, T. Richter, C. Lessig, Deep neural networks for geometric multigrid methods, https://doi.org/10.48550/ARXIV.2106.07687, 2021.

[31] N. Margenberg, R. Jendersie, C. Lessig, T. Richter, DNN-MG: a hybrid neural network/finite element method with applications to 3D simulations of the Navier-Stokes equations, arXiv:2307.14837 [physics], Jul. 2023, https://doi.org/10.48550/arXiv.2307.14837, http://arxiv.org/abs/2307.14837.

[32] D. Ray, J.S. Hesthaven, Detecting troubled-cells on two-dimensional unstructured grids using a neural network, J. Comput. Phys. 397 (2019) 108845, https://doi.org/10.1016/j.jcp.2019.07.043.

[33] K. Carlberg, V. Forstall, R. Tuminaro, Krylov-subspace recycling via the POD-augmented conjugate-gradient method, SIAM J. Matrix Anal. Appl. 37 (3) (2016) 1304–1336, https://doi.org/10.1137/16M1057693.

[34] D. Pasetto, M. Ferronato, M. Putti, A reduced order model-based preconditioner for the efficient solution of transient diffusion equations, Int. J. Numer. Methods Eng. 109 (8) (2017) 1159–1179, https://doi.org/10.1002/nme.5320.

[35] A. Kaneda, O. Akar, J. Chen, V. Kala, D. Hyde, J. Teran, A deep conjugate direction method for iteratively solving linear systems, arXiv:2205.10763 [cs, math], Oct. 2022, https://doi.org/10.48550/arXiv.2205.10763.

[36] A. Heinlein, A. Klawonn, M. Lanser, J. Weber, Machine learning in adaptive domain decomposition methods—predicting the geometric location of constraints, SIAM J. Sci. Comput. 41 (6) (2019) A3887–A3912, https://doi.org/10.1137/18M1205364.

[37] A. Taghibakhshi, N. Nytko, T.U. Zaman, S. MacLachlan, L. Olson, M. West, MG-GNN: multigrid graph neural networks for learning multilevel domain decomposition methods, arXiv:2301.11378 [cs, math], Mar. 2023, https://doi.org/10.48550/arXiv.2301.11378.

[38] A. Grebhahn, C. Rodrigo, N. Siegmund, F.J. Gaspar, S. Apel, Performance-influence models of multigrid methods: a case study on triangular grids, Concurr. Comput. Pract. Exp. 29 (17) (2017) e4057, https://doi.org/10.1002/cpe.4057.

[39] P.F. Antonietti, M. Caldana, L. Dede', Accelerating algebraic multigrid methods via artificial neural networks, Vietnam J. Math. 51 (1) (2023) 1–36, https://doi.org/10.1007/s10013-022-00597-w.
[40] I. Luz, M. Galun, H. Maron, R. Basri, I. Yavneh, Learning algebraic multigrid using graph neural networks, arXiv:2003.05744 [cs, stat], Sep. 2020, https://doi.org/10.48550/arXiv.2003.05744.
[41] N.S. Moore, E.C. Cyr, C.M. Siefert, Learning an Algebriac Multrigrid Interpolation Operator Using a Modified GraphNet Architecture, Tech. Rep. SAND2022-3579, Sandia National Lab. (SNL-NM), Albuquerque, NM (United States), Nov. 2021.
[42] Y. Azulay, E. Treister, Multigrid-augmented deep learning preconditioners for the Helmholtz equation, arXiv:2203.11025 [cs, math], Mar. 2022, https://doi.org/10.48550/arXiv.2203.11025.
[43] S. Bhowmick, V. Eijkhout, Y. Freund, E. Fuentes, D. Keyes, Application of machine learning to the selection of sparse linear solvers, Int. J. High Perform. Comput. Appl. (2006).
[44] S. Bhowmick, V. Eijkhout, Y. Freund, E. Fuentes, D. Keyes, Application of alternating decision trees in selecting sparse linear solvers, in: Software Automatic Tuning, Springer, 2011, pp. 153–173.
[45] S. Bhowmick, B. Toth, P. Raghavan, Towards low-cost, high-accuracy classifiers for linear solver selection, in: International Conference on Computational Science, Springer, 2009, pp. 463–472.
[46] E. Fuentes, Statistical and Machine Learning Techniques Applied to Algorithm Selection for Solving Sparse Linear Systems, Doctoral Dissertations, Dec. 2007, https://trace.tennessee.edu/utk_graddiss/171.
[47] P. Motter, K. Sood, E. Jessup, B. Norris, Lighthouse: an automated solver selection tool, in: Proceedings of the 3rd International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering, 2015, pp. 16–24.
[48] A. Holloway, T.-Y. Chen, Neural networks for predicting the behavior of preconditioned iterative solvers, in: Y. Shi, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2007, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 302–309.
[49] A. Odot, R. Haferssas, S. Cotin, Deepphysics: a physics aware deep learning framework for real-time simulation, arXiv:2109.09491, 2021.
[50] I.B. Yahia, J.-P. Merlet, Y. Papegay, Mixing neural networks and the Newton method for the kinematics of simple cable-driven parallel robots with sagging cables, in: 2021 20th International Conference on Advanced Robotics (ICAR), 2021, pp. 241–246.
[51] A. Lechevallier, S. Desroziers, T. Faney, E. Flauraud, F. Nataf, Hybrid Newton method for the acceleration of well event handling in the simulation of Co2 storage using supervised learning, https://doi.org/10.2139/ssrn.4696416, Jan. 2024, https://papers.ssrn.com/abstract=4696416.
[52] L. Luo, X.-C. Cai, PIN-L: preconditioned inexact Newton with learning capability for nonlinear system of equations, SIAM J. Sci. Comput. 45 (2) (2023) A849–A871, https://doi.org/10.1137/22M1507942.
[53] V.L. Silva, P. Salinas, M. Jackson, C. Pain, Machine learning acceleration for nonlinear solvers applied to multiphase porous media flow, Comput. Methods Appl. Mech. Eng. 384 (2021) 113989, https://doi.org/10.1016/j.cma.2021.113989.
[54] A. Zandbergen, Master thesis, Delft University of Technology, 2022, https://repository.tudelft.nl/islandora/object/uuid%3Ab3343bea-1270-4280-8192-7f08d162fbb0.
[55] P. Wesseling, Elements of Computational Fluid Dynamics, TU Delft, Delft, 2014.
[56] F.P. Incropera, D.P. de Witt, Fundamentals of Heat and Mass Transfer, 2nd edition, Wiley, New York, 1985.
[57] COMSOL Multiphysics® v.6. 1, COMSOL AB, Stockholm, Sweden, COMSOL Multiphysics Reference Manual, https://doc.comsol.com/6.1/doc/com.comsol.help.comsol/COMSOL_ReferenceManual.pdf, 2022.
[58] G. Hauke, T. Hughes, A unified approach to compressible and incompressible flows, Comput. Methods Appl. Mech. Eng. 113 (3) (1994) 389–395, https://doi.org/10.1016/0045-7825(94)90055-8.
[59] T.J. Hughes, M. Mallet, A new finite element formulation for computational fluid dynamics: III. The generalized streamline operator for multidimensional advective-diffusive systems, Comput. Methods Appl. Mech. Eng. 58 (3) (1986) 305–328, https://doi.org/10.1016/0045-7825(86)90152-0.
[60] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (4) (1989) 303–314, https://doi.org/10.1007/BF02551274.
[61] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, https://doi.org/10.48550/arXiv.1412.6980, Jan. 2017.
[62] H.J. Kelley, Gradient theory of optimal flight paths, ARS J. 30 (10) (1960) 947–954.
[63] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, Adaptive Computation and Machine Learning, MIT Press, Cambridge, MA, 2016.
[64] E. Philip, W. Murray, M.A. Saunders, User's guide for snopt version 7: software for large-scale nonlinear programming, 2015.
[65] P.E. Gill, W. Murray, M.A. Saunders, Snopt: an sqp algorithm for large-scale constrained optimization, SIAM J. Optim. 12 (4) (2002) 979–1006, https://doi.org/10.1137/S1052623499350013.
[66] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366, https://doi.org/10.1016/0893-6080(89)90020-8.
[67] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netw. 4 (2) (1991) 251–257, https://doi.org/10.1016/0893-6080(91)90009-T.
[68] E. Kreyszig, Advanced Engineering Mathematics, John Wiley & Sons, 2010, https://books.google.de/books?id=UnN8DpXI74EC.
[69] T.S. Cohen, M. Welling, Group equivariant convolutional networks, arXiv:1602.07576 [cs, stat], Jun. 2016, https://doi.org/10.48550/arXiv.1602.07576, http://arxiv.org/abs/1602.07576.
[70] D.G. Anderson, Iterative procedures for nonlinear integral equations, J. ACM (JACM) 12 (4) (1965) 547–560.