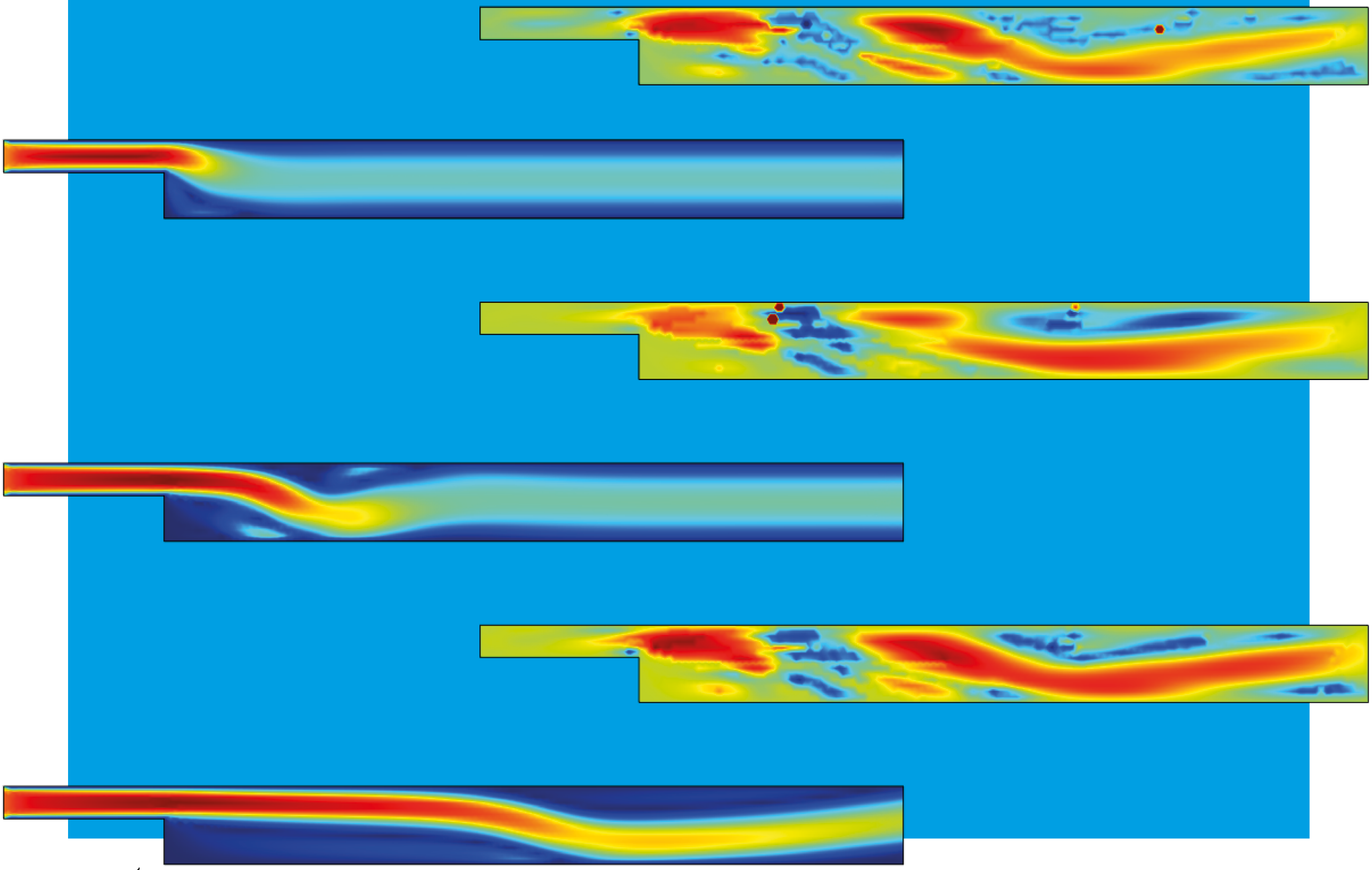
Predicting the optimal CFL number for pseudo time-stepping

with machine learning in the COMSOL CFD module

Anouk Zandbergen



TU Delft

COMSOL

Predicting the optimal CFL number for pseudo time-stepping

with Machine Learning in the COMSOL CFD module

by

Anouk Zandbergen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday September 28, 2022 at 13:30.

Student number:	4594967	
Project duration:	November 29, 2021 – September 28, 2022	
Thesis committee:	Dr. M. Möller,	TU Delft
	Dr. A. Heinlein,	TU Delft, supervisor
	Dr. D.J.P. Lahaye,	TU Delft
	Dr. T.L. van Noorden,	COMSOL, supervisor

Cover Image: solutions of backwards facing step simulation in COMSOL with their corresponding optimised CFL number.

TUDelft

Abstract

Commercial finite element software like COMSOL is build to be user-friendly. For example, the user does not have to find weak formulations, or discretise the partial differential equations by hand. One of the more difficult parts of using finite element software is deciding which solver method and solver parameters to choose, as the performance of the solver, and hence the performance of the whole simulation, depends on it.

Pseudo time-stepping is a stabilisation method that is designed to lead to a stationary solution for a wider range of initial guesses compared to traditional Newton methods. During pseudo time-stepping, a CFL number is used. This CFL number is adapted each nonlinear iteration. In COMSOL the CFL number either depends on the iteration count, or the nonlinear error estimate and previous CFL number. In this case, the CFL number is a global value, but the pseudo time-step itself is local since it also depends on the mesh cell size.

In this research, a neural network is created to predict a local CFL number for pseudo time-stepping, such that the network accelerates convergence compared to the two CFL numbers for pseudo time-stepping in COMSOL and is generalisable. The convergence is assumed to be accelerated if the network predictions result in convergence in fewer nonlinear iterations compared to solvers that use each one of the two CFL numbers in COMSOL. The network is assumed to be generalisable if it is able to accelerate the convergence for different laminar flow problems in COMSOL on which the network has not been trained.

A network with 2 hidden layers with each 16 neurons is trained on local data from element patches in order to make it generalizable. The element patch data points contain information of the central triangular element vertices, and of the vertices of the three adjacent elements. The local data consist of the velocities, pressure and residuals, as well as the cell Reynolds number and the element edge lengths. The loss of this network is the root mean squared error between the network output and a previously computed CFL target. This target is the optimised value for the local CFL number such that the difference between the solution obtained from the pseudo time-step and the exact solution is minimised.

The network predictions were able to accelerate convergence compared to to the two CFL numbers in COMSOL for several different 2D laminar flow simulations. Therefore, the conclusion is that a network trained on optimised local CFL numbers is able to generalize well and accelerate the existing pseudo time-stepping method.

Preface

This master project is written to obtain the degree of Master of Science Applied Mathematics at the Delft University of Technology. As this work focuses on neural networks that predict nonlinear solver parameters, the topic of this project combines my interest in numerical analysis and machine learning.
Before starting my thesis in collaboration with COMSOL, I did an internship there. This work made me enthusiastic, which made me decide to expand the research performed for the internship into a master thesis. With joy I worked at COMSOL for over a year, and worked together with my two supervisors Alexander Heinlein (TU Delft) and Tycho van Noorden (COMSOL). Thank you both for the advice and enthusiasm for the project.

I want to dedicate this work to grandma. Lieve oma, wat fijn en bijzonder dat wij dit samen nog kunnen meemaken.

Anouk Zandbergen
Delft, September 2022

Contents

1 Introduction

The Finite Element Method (FEM) is a popular numerical method for discretising differential equations related to engineering and modelling problems. Software that uses the FEM to solve problems in the fields of, for example, structural analysis and fluid flows are popular among engineers. Within these software packages it is possible to build models and perform simulations, where the difficult discretisation and solving phases are performed automatically by the software itself, making it user-friendly. There is a wide range of different FEM software, of which COMSOL Multiphysics is one [5].

One challenge, however, is the fact that the performance of the solver depends on the solver method and solver parameters. And there are a lot of nonlinear solver methods, such as Automatic Newton and Newton Constant, and a lot of solver parameters, such as damping factor of the nonlinear solver and which initial conditions to use. Standard solver settings are available in COMSOL, so that the software user does not have to think about them. However, the standard solver settings are not always the best settings for a given problem. In the worst case the solver will not converge with the standard settings, but can converge and give a solution with the right solver settings. Choosing the right solver settings is a challenge, also for experienced engineers, physicists and mathematicians.

In this project, the CFL number for pseudo time-stepping will be considered for a neural network to predict. Pseudo time-stepping is a method for solving stationary nonlinear equations. The pseudo time-step depends on the CFL number, the solution vector and the mesh cell size, making the pseudo time-step a local value. The CFL number computed in COMSOL either depends on the nonlinear iteration count, or the nonlinear error estimate. Therefore, the CFL number is a global value. Thus, it is interesting to investigate if a neural network can predict a locally optimal CFL number, that depends on certain solution parameters, and by doing so, improve the pseudo time-step method.

The aim of the thesis is to answer the question: "*can a neural network be trained to predict the local CFL number for pseudo time-stepping, such that it accelerates the convergence and is generalisable*?". Convergence is accelerated if the CFL prediction of the network result in convergence in less nonlinear iterations compared to pseudo time-stepping with one of the two CFL numbers given in COMSOL. Generalisable means that the network is able to accelerate convergence for several different 2D laminar flow simulations.

For the network, two different loss functions are considered. The loss function that is best able to train the network to predict optimal CFL numbers will be used. In order to achieve generalisability, local data from element vertices and centroids are used to train the network in Matlab. The information from the vertices are the velocities, the pressure and the residuals, and also the cell Reynolds number and element edge lengths are used as an input. Model specific information is thus not used, resulting in a network that can be applied to different laminar flow simulations. Furthermore, the data is obtained from different back-step flow models in COMSOL. Each back-step has a different geometry, Reynolds number, velocity and mesh size, in order to create much variety within the training data.

The structure of the thesis is as follows: first related work is discussed in section 2. Second, in section 3 necessary background information about Navier-Stokes, pseudo time-stepping and neural networks is given. Then, in section 4, the loss functions, the data sets and neural networks for CFL prediction are explained. The results of the network are given in section 5. Lastly, the conclusion and recommendations are given in section 6.

In the appendix, the different considered laminar flow models in this research are given in A. In B, sensitivity analysis and optimisation study steps in COMSOL required to compute the loss functions is given. Furthermore, results from using patches as input data or data from single elements is discussed in C. The tables with all the results from the network are given in D. Finally, some of the important and useful code used with Matlab livelink is given in E.

2 Related work

Using machine learning and neural networks to replace or enhance existing numerical methods is currently a popular research topic. However, there is not a lot of literature available of research on predicting the optimal solver settings with machine learning. In this research we are especially interested in using local data for training a neural network in order to obtain generalisability.

Four papers ([17, 18, 22, 23]), that use local data to train machine learning models to predict solver parameters, will be discussed. In [17, 18, 22] neural networks are used, while in [23] a random forest turned out to be the best option for making predictions. Generalisability is discussed in all these papers and will also be discussed here. To the best of our knowledge, these are the only papers that use local data to predict solver parameters.

Secondly, papers ([1, 2, 3, 7, 11, 12, 20]) that use information of discretisation matrices of linear systems to predict linear solvers will also be discussed. Prior to the graduation project, the effectiveness of neural networks for predicting solvers using global data has been investigated during a three-month internship at COMSOL. The results of these papers, as well as the results from the internship, will be discussed.

Finally, we will discuss how the research in the papers relate to the graduation project, what the differences are and which techniques can be and will be used for this research.

2.1. Predicting solver parameters using machine learning

Predicting solver parameters with the use of a neural network trained on local data has been investigated by Margenberg et al. in [17, 18], Ray and Hesthaven in [22], and by Silva et al. in [23]. First the papers [17, 18] are discussed, followed by respectively [22] and [23].

In both [17] and [18] a network is created that enhances multigrid solvers by letting the network correct interpolated solutions on the fine grids. The network uses information from patches of elements. So, not only local data from one element on which the networks makes a prediction is considered for the input, but also local information of the adjacent elements are used. The patches give more information compared to only one element, and the local information is used for generalisability. In other words, to make the network applicable to different flow problems. In [18] the authors performed additional research on the network they created in [17]. We will first discuss their initial research [17].

First, the Navier-Stokes equations are solved for the first “coarse” layers of the multigrid hierarchy. Then the network predicts a correction of the interpolated solution vector towards the real and unknown solution. This correction is then feed back to the first “coarse” layers and affects the time evolution of the solution, without using the expensive “fine” multigrid layer hierarchy [15].

For training this network, local information of the discretised problem is used as an input. That is the residuals, velocities, cell size and Péclet number. With this local information, the loss is then computed as the difference of the solution that uses the correction output of the network and a reference solution. This reference solution is obtained by actually performing the multigrid method with levels of finer grids.

Furthermore, the network memorizes the past of the flow. Such networks are called recurrent neural networks (RNN) [9], and this type of networks have an additional activation which are discrete in time.

The network reduces the computation time, while obtaining the same or higher accuracy compared to standard multigrid solvers for some of the models in [17]. However, not all of the models showed improvements, thus further research is needed to improve the generalisability of this network.

In [18] Margenberg et al. investigates how the size of the network influences the performance and the training of the network. They mostly discuss the sizes regarding the Gated Recurrent Units (GRU), a type of RNN [9], and the hidden state size of the GRU-cells.

The GRU-cells are layers of the neural network that memorize the output of the previous time step and uses it as an input for the current time step [9]. These cells can be stacked on to each other. The size of the hidden states is the number of memorized time steps, so when the size is 16 the network will remember outputs of the previous 16 iterations [9]. The paper [18] compares results of networks that use different sizes for the hidden states and GRUs stacked on to each other.

Results of networks with only 1 GRU, and 2 or 3 GRU-cells stacked on to each other with either a hidden state size of 32 or 64. The network with GRU dimension of 64×3 performed best: it was able to improve the accuracy of the standard solver without adding extra computational costs. Therefore, using an RNN for predicting solver parameters could also be an interesting and effective approach for this research. Unfortunately, in both [17] and [18], the dimensions of the neural network in terms of hidden layers and number of neurons per layer is not discussed.

In [22] a network is used to detect troubled-cells in a mesh. These troubled-cells suffer from Gibbs oscillations as a result of applying high-order numerical methods in regions with discontinuities. Limiting the solution during post-processing can control these oscillations, but these methods also limit in cells that do not show oscillations, making it computationally expensive and reduces the accuracy in smooth regions. Detecting the troubled-cells and only limit in these can increase the accuracy and is less computationally expensive.

There already are methods to flag troubled-cells, such as minmon-type TVB (total variation bounded) limiters [22]. However, these methods require problem-dependent parameters. These parameters are determined empirically, and a poor choice of the parameters result in a poor result of the troubled-cell detection. Training a neural network to detect these cell does not require these empirically determined problem-dependent parameters.

The input data of the neural network contains local information of the cells, which results in a generalisable network. The outcome classifies the cells into the classes “good” or “troubled’. The used loss function is Cross-entropy, which is often used to optimise classification models [9]. The targets used are classified troubled cells, obtained by labelling the troubled cell one by one. The neural network has 5 hidden layers with each 20 neurons.

This method that uses the network outperforms the traditional TVB indicator in terms of solution accuracy and number of flagged cells. The computational costs of these methods are comparable. This research is performed with a structured quadrilateral two-dimensional grid.

In [23] the authors investigate if a machine learning method can predict a relaxation parameter to accelerate any type of numerical relaxation based nonlinear solver. This nonlinear solver is applied to multiphase porous media flow. The used nonlinear solver in this paper is a Picard method, that has three main loops. A relaxation parameter ω was added to accelerate the convergence. An initial guess ω^0 for this method is required: it has to be small enough to avoid divergence, but at the same time also as large as possible to be able to accelerate the convergence. The authors created two machine learning models to predict ω^0 based on physics of the model in order to accelerate the nonlinear solver.

The two approaches both use dimensionless numbers and simulation properties as an input for the machine learning model. Both use the mean squared error between the predicted relaxation parameter and the optimised relaxation parameter. The first approach, however, also uses the number of inner nonlinear iterations at the outer nonlinear iteration as an input, and outputs the relaxation parameter ω^0. The second approach uses these parameters the other way around: ω^0 is added as an input and the number of inner nonlinear iterations at the outer nonlinear iteration is the output. The consequence for the second approach is that an extra optimisation step is required to find the best relaxation parameter.

These inputs are not local, they instead use dimensionless numbers and simulation properties to achieve generalisability. By using these inputs, the authors state they can use a simple two-dimensional layered reservoir to generate the training data, while exploring a wide range of the parameter space.

In total 6500 simulations of the reservoir model were run with varying porosities, horizontal and vertical permeabilities, time steps, gravity magnitude and direction, and relaxation factor to create the data set.

They state that the best performing machine learning method for this problem was random forest regression [9]. For the two methods, a grid search was performed on the two random forests. Both random forests did not use bootstrap, had 50 trees in the forest, had a maximum tree depth of 30 and had a minimum of 2 samples to be a leaf node. The only difference between the two forest architectures was in the numbers of features to be considered for splitting: 10 for approach 1 and 5 for approach 2.

The first method outperformed the second method in both accuracy and steps required to obtain the optimal relaxation parameter ω^0. The performance of the solver has been increased by more than 24% with the use of this random forest, while also increasing the robustness of the solver. These results have been obtained by applying the model on "*more realistic and challenging reservoir models*" [23], showing its generalisability.

2.2. Selecting numerical solvers for linear systems using machine learning

Not only solver parameter optimisation with machine learning is studied, also solver selection for linear numerical systems with machine learning is studied. Using information from the discretisation matrix, a machine learning model is created to predict the optimal numerical method to solve the system. This is studied by, Bhowmick et al. in [1, 2, 3], Grebhahn et al. in [11], Holloway and Chen in [12], Motter et al. in [20], and by Fuentes in [7].

In the papers of Bhowmick et al. [1, 2, 3], using machine learning methods to select (sparse) linear solvers is discussed. In all papers, matrix properties are used. These properties can be categorized in structural properties, norms, estimated spectral properties, normality and variance. Examples of matrix properties are trace (in absolute value), number of rows, different matrix norms, bandwidths and estimated condition number. In total 57 features are given to the machine learning model. In the first paper [3], they explain how machine learning can be applied for the selection of solvers for sparse linear systems. This problem is formulated as a classification problem, so the output of the network can either be classified as "good" or "bad". The investigated machine learning methods are boosting and an Alternating Decision Tree (ADT) [9]. In boosting predictors are trained to correct the mistakes of the previous trained predictor [9]. In the second paper [1], the K-nearest ($NN(k)$) algorithm, the ADT, naive Bayes and the Support Vector Machine (SVM) are discussed [9]. In order to obtain low-cost, high-accuracy classifiers, they order the features in increasing order of their computational complexity. In [2], an ADT is used to predict the efficient solver by using information of the sparse linear systems and so-called runtime-dependent features. These runtime-dependent features are, for example, the stages of simulation.

The authors of [11] create a performance-influence model that predicts the optimal configuration options when using a multigrid solver. These configuration options include the smoother, the construction of coarser levels and discrete problem considered on them, the inter-grid transfer operators, the cycle type and the number of pre- and post-smoothing steps [11]. There are no general rules to choose these parameters and direct measurement of all these variants is computationally expensive. At the same time the performance-influence model was able to predict the performance of all variants with an accuracy of 95.5%, while measuring less than 1% of the variants. That means using the performance-influence model is an efficient and effective method to choose the optimal parameters for multigrid solvers.

In [12], the effectiveness of neural networks as a machine learning method to predict if a given combination of preconditioner and iterative method will be able to solve a given sparse linear system is investigated. The network is trained on matrices that are ran on a preconditioner with certain combination of parameters. The trained network then has to classify the sparse linear system into solvable or not solvable. The correctly classified systems ranged from 73.3% to 98.6% for the three different cases investigated.

In [20], a machine learning model is created to find the most suitable numerical algorithm for a given problem. It uses information of the discretisation matrix of large linear systems that the user aims to solve. The Lighthouse machine learning model is based on a decision tree, where the inputs are the features of the matrix and the solution time when that discretisation matrix is used. The decision tree then labeled methods within 35% of the best solver time as a good decision. In [20] it is stated that using Lighthouse, the productivity of the developer and the performance of the application can be improved. In [7], neural networks are also used to predict an appropriate numerical solver for sparse linear systems.

The papers mentioned all investigate how machine learning can be implemented to predict the best linear solver by using data from the sparse linear system matrix, and at the same time which machine learning method can be applied best for this problem. Regardless of the machine learning method used, the described input data and methodologies are effective for the described problems.

Unfortunately, it is not mentioned which hyper parameters are used for these machine learning models. So, for example, when neural networks are used for the solver prediction, it is thus not known for the reader what number of hidden layers and neurons per hidden layer is used.

2.3. Selecting non-linear solvers using neural networks

During a three-month internship at COMSOL, the effectiveness of neural networks for predicting non-linear solvers using global data of a back-step simulation has been investigated. The inflow velocity, the geometry and the initial velocity of the COMSOL simulation were varied and information of the convergence of four possible nonlinear solvers in COMSOL was available. For each of the four nonlinear solvers, a neural network was created to predict if this solver would lead to convergence and in how many iterations.

These four neural networks have four hidden layers with 128 neurons per layer. Each network predicts if the nonlinear solver would converge and approximates the required number of nonlinear iterations for convergence. If a solver had the label that it would result in convergence with the least amount of iterations, the network would choose that as the preferred nonlinear solver for the given back-step simulation.

In 66% of the cases this model was able to predict the optimal solver correctly. If the default nonlinear solver in COMSOL was always used for each back-step, only in 44% the optimal solver was used. Thus, for the back-step simulations, by using the neural network model instead of the default solver settings, it is more likely that the optimal solver for that simulation is used.

A downside of this model is that it can only be applied to back-step models, i.e. it is not generalisable. This was a motivation to perform research in machine learning models that are applicable to more simulations during this graduation project.

2.4. Comparison of methods from the literature

We have discussed four papers regarding machine learning that use local data to predict optimal solver parameters, seven papers that use information of linear systems to predict the optimal linear solver and an internship research about using neural networks to predict the optimal nonlinear solver for a given back-step model.

The aim of this research is to predict the CFL number for pseudo time-stepping such that the COMSOL simulation converges in the least amount of iterations. This CFL number is a solver parameter of the pseudo time-stepping algorithm. Therefore, the first four papers by Margenberg et al. [17, 18], Ray and Hesthaven [22] and Silva et al. [23] are the most interesting for this research.

All these four papers use local data to train a machine learning model. They state that using local data can enhance the generalisability of the model, which is also shown in their results in [17, 18, 22]. But also generalisability is achieved in [23], where dimensionless numbers and simulation properties are used instead. In these papers, generalisability is achieved by using a recurrent neural network in [17, 18], a neural network in [22] and a random forest in [23]. In this research, the effectiveness of a neural network to predict the CFL number for pseudo time-stepping using local data will be investigated.

All these papers also use a target for training the network. In [17, 18], they use the obtained solution of the finer grid hierarchy as a reference solution for training. In [22], they already determined the troubled cells using a computationally expensive method. Finally, in [23], the loss function is the mean square error of the prediction and the optimised relaxation parameter.

Lastly, all the described machine learning methods enhance existing numerical methods by optimising certain solver parameters. Thus, these machine learning models do not completely replace these numerical methods.

In this research, we will use local data for generalisability and a neural network to predict the optimal CFL value. Like papers [17, 18, 22, 23], a loss with a training target will be used. This training target is a previously computed optimal CFL number. Also a second loss is considered. Unlike the loss functions described in the literature, this loss does not use a target. The network predictions are directly used to compute one pseudo-time step iteration in COMSOL, and the value of an error estimate will then serve as the loss. This method of using the FEM software to compute the loss and its gradients has not been used in the literature yet.

Also, in this research information of patch elements is used, like in [17, 18, 22]. In the patch, information of the central element and the adjacent elements are used to generate input data.

Furthermore, using a recurrent neural network can be an interesting extension, as its effectiveness is shown in both papers [17, 18]. A recurrent neural network is able to analyse time series data and can possibly anticipate on future predictions, as explained in [9]. Since the CFL numbers in COMSOL are either computed by the nonlinear iteration count, or by the error estimate and previous CFL number, information of future and previous CFL numbers can improve the network prediction. This is an interesting direction for future research and will also be discussed in section 6.1.

3
Background information

In this section, first a short introduction to the Navier-Stokes equations is given. The simulations considered in COMSOL are laminar flow simulations, described by these stationary Navier-Stokes equations. These equations are discretised and solved in COMSOL.

Secondly, the pseudo time-step algorithm and the definition of the CFL number that plays a role in this algorithm are given. We will explain why investigating and predicting local CFL numbers can enhance the pseudo time-stepping method.

A neural network is used to predict local CFL numbers for pseudo time-stepping. Necessary information about neural networks is given. Here, the multi-layer perceptron and backpropagation will be explained.

3.1. Navier-Stokes equations

The Navier-Stokes equations are nonlinear partial differential equations that describe viscous flow. In the problem with the back-step simulation, we consider incompressible flow. That is a flow in which the density of each material particle remains the same [26]. The stationary Navier-Stokes equations for incompressible flow are given by

$$\begin{aligned} \rho(\mathbf{u}\cdot\nabla)\mathbf{u}-\mu\nabla^2\mathbf{u} &= -\nabla p+\mathbf{f}, \\ \rho\nabla\cdot\mathbf{u} &= 0. \end{aligned} \tag{3.1}$$

In these equations, $\mathbf{u}$ is the flow velocity, μ is the 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 [26].

In COMSOL, we use the laminar flow physics interface. A flow remains laminar as long as the Reynolds number is below a critical value in the order of 10^5 [13]. The Reynolds number Re is given by

$$\text{Re} = \frac{\rho U L}{\mu}. \tag{3.2}$$

Here, U and L are the typical velocity and length scale. The Reynolds number gives the ratio between inertial and viscous forces [8]. 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. The cell Reynolds numbers in the back-step flow models that are used to create a data set for training a neural network have values between 0 and 400.

3.2. Spatial discretisation

In COMSOL, the stationary Navier-Stokes equations are discretised by the finite element method. The geometry of the flow problem is then subdivided into elements. The elements used for this thesis are triangular. After the finite element method has been applied to discretise the stationary Navier-Stokes equations we write $F(\tilde{\mathbf{u}}) = 0$ to represent the system of nonlinear equations.

Algorithm 1 Pseudo time-stepping

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

Since the stationary Navier-Stokes equations are nonlinear, a numerical method that solves nonlinear equations must also be applied. In COMSOL there are several numerical methods that can be chosen to solve nonlinear equations. The default is automatic Newton, where the damping factors are automatically adapted each nonlinear iteration. It is also possible to apply Newton with a fixed damping factor, called Newton constant.

3.3. Pseudo time-stepping

Solving stationary nonlinear equations with Newtons method requires an initial guess close enough to the root. Globalization strategies are able to give a result for a wider range of initial guesses, but can stagnate at local minima [14].

Pseudo time-stepping, or pseudo-transient continuation, is an alternative to compute stationary solutions for partial differential equations. It is designed to obtain a solution more often by using the PDE structure of the given problem [14]. The method frames the problem in a time-depending setting:

$$\frac{\partial \tilde{\mathbf{u}}}{\partial t} = F(\tilde{\mathbf{u}}). \tag{3.3}$$

with $F(\tilde{\mathbf{u}}) = 0$ the system of nonlinear equations of which a root must be determined, and $\tilde{\mathbf{u}}$ the spatial discretisation of $\mathbf{u}$. In [14] the algorithm for pseudo time-stepping is described as follows

$$\frac{\partial \tilde{\mathbf{u}}}{\partial t} = -V^{-1}F(\tilde{\mathbf{u}}), \quad \tilde{\mathbf{u}}(0) = \tilde{\mathbf{u}}_0, \tag{3.4}$$

where V is a nonsingular matrix that scales the problem. The sequence of iterations is given by

$$\tilde{\mathbf{u}}_{n+1} = \tilde{\mathbf{u}}_n - \left(V + F'(\tilde{\mathbf{u}}_n)\right)^{-1} F(\tilde{\mathbf{u}}_n), \tag{3.5}$$

with $F'(\tilde{\mathbf{u}}_n)$ the Jacobian. The pseudo code from [14] is given in algorithm 1. In COMSOL [8] the nonsingular matrix V^{-1} is given by the local pseudo time-step, defined as

$$V^{-1} = \Delta\tilde{t}_n = \text{CFL}(n)\frac{h}{||\mathbf{u}||}, \tag{3.6}$$

with h the mesh cell size and CFL(n) the CFL number used for pseudo time-stepping in COMSOL [8]. The CFL number is not locally defined, but is the same for the whole geometry. The pseudo time-step $\Delta\tilde{t}$, however, is a local value since h depends on the size of each mesh element and $||\mathbf{u}||$ on the solution on each mesh cell.

There are two options for the CFL number within COMSOL. In the first option, the local CFL number used in COMSOL depends on the iteration count n of the nonlinear solver:

$$\text{CFL}(n) = 1.3^{\min(n,9)} + \text{if}(n > 20, 9 \cdot 1.3^{\min(n-20,9)}, 0) + \text{if}(n > 40, 90 \cdot 1.3^{\min(n-40,9)}, 0). \tag{3.7}$$

In the second option, the CFL number depends on the nonlinear error estimate e_n for iteration n, the the given target error estimate “tol” and the controller parameters k_P, k_I and k_D, which are positive constants:

$$\text{CFL}(n+1) = \left(\frac{e_{n-1}}{e_n}\right)^{k_P} \left(\frac{\text{tol}}{e_n}\right)^{k_I} \left(\frac{e_{n-1}/e_n}{e_{n-2}/e_{n-1}}\right)^{k_D} \text{CFL}(n). \tag{3.8}$$

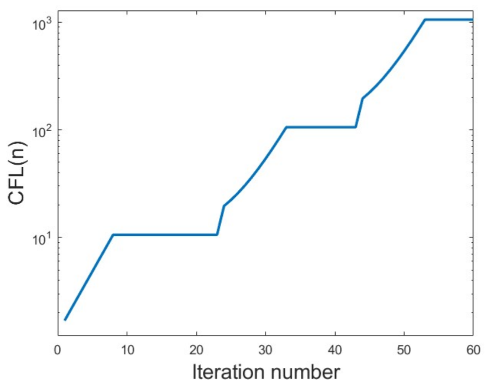


Figure 3.1: A plof of CFL(n) that depends on the nonlinear iteration count.

For this CFL number a hard lower limit CFL(n) ≥ 1 is used and convergence is accepted when CFL(n) $\geq$ CFL$_\infty = 10^4$.

For both CFL numbers, in the initial iterations of pseudo time-stepping, the method starts with a small CFL number and gradually increases it as the solution reaches convergence [8]. In (3.7) this is because the CFL number increases each iteration, also shown in figure 3.1. For (3.8), when the solution is near convergence, the error estimates e_n will be smaller and thus the CFL number increases. When the pseudo time-step approaches infinity, the method will behave as a normal Newton step.

As we can see in equations (3.7) and (3.8), the CFL number is a global expression as it is the same for each mesh element. The pseudo time-step, however, does depend on the size of the mesh cell and solution on the mesh cell, making the time-step itself a local expression.

Since the local CFL number either depends on the number of iterations, or on the global error estimate, it is interesting to investigate if a CFL number that varies per element performs better. For example, elements with high local residuals could then have smaller local CFL numbers compared to elements with low residuals. Since the pseudo time-step behaves like a Newton step when it is very high, it can accelerate convergence in areas which already have low residuals. So, treating each mesh element individually using their local information can accelerate convergence for the whole simulation.

One could think of adding local information about the solution, residuals, mesh and the cell Reynolds number. A neural network can have all this information as a input and can then output a CFL number for each mesh cell, resulting in a local CFL number that takes all this information into account.

Within this thesis, the whole pseudo time-step $\Delta\tilde{t}_n$ in (3.6) will be replaced by a network prediction. We will still call this the CFL prediction, but in fact it is the whole nonsingular matrix $\Delta\tilde{t}_n$ that will be predicted. To be able to do that in COMSOL, a weak contribution is formulated that replaces $\Delta\tilde{t}_n$ with the network prediction. More about the weak formulation and pseudo time-stepping in COMSOL is given in appendix A.

3.4. Neural networks

An artificial neural network is a deep learning method that is inspired by how our own human nervous system processes information and has been introduced around the 1940s [9, 25]. The interest of using and investigating neural networks has been up and down. Improvements of the network architectures led to an increasing interest in the early 1980s, but when other alternative machine learning methods, like Support Vector Machines (SVM), were created which seemed to perform better, there was a decreased interest in the 1990s [9]. Right now, neural networks are again a hot research topic in several fields of engineering, mathematics and computer science. In [9] some reasons are summarised:

- Availability of a huge quantity of data, where neural network mostly outperform other machine learning techniques;
- Increased computing power due to improved GPU cards and Moore's Law, resulting in a reasonable training time for the neural network;

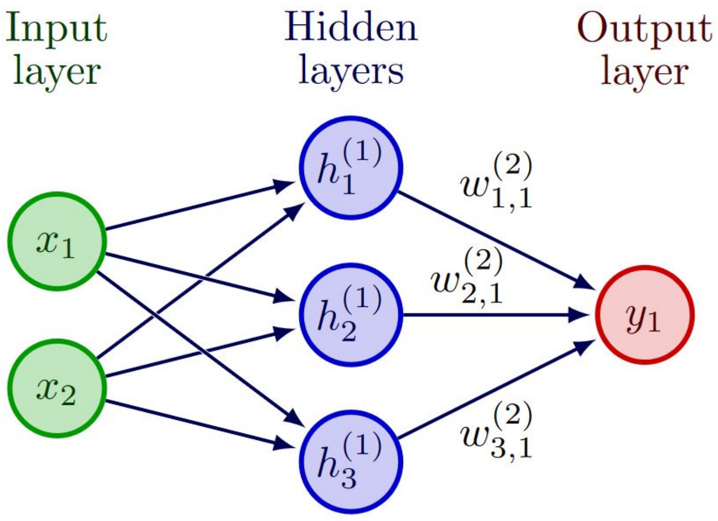


Figure 3.2: An MLP with two input neurons, three neurons in the hidden layer and one output neuron. The weights between the hidden layer and the output layer are shown, but are left away between the input and hidden layer for simplicity.

- Improved training algorithms for neural networks;
- There is more attention to successful neural network studies, resulting in more interest and curiosity to investigate the possibilities of neural networks;
- Availability of software and manuals that make programming, creating and using neural networks relatively easy.

In this thesis, a neural network will be created to investigate if it can predict the optimal CFL number for pseudo time-stepping, such that it accelerates convergence and generalises. That is, obtaining a converged solution with less nonlinear iterations compared to the COMSOL default for pseudo time-stepping and the network is able to accelerate convergence for different laminar flow simulations.

3.4.1. Multi-layer perceptron

A neural network has an input layer, hidden layers and an output layer with in each layer a number of neurons. The depth of the network is the amount of hidden layers that the network contains, and the width is the amount of neurons per hidden layer.

The network we will build has multiple hidden layers and is therefore a Mmlti-layer perceptron (MLP). An MLP has an input layer that passes through the information to the first hidden layer. If the network has two or more hidden layers, it is often referred to as a deep neural network or DNN. Finally, the last hidden layer passes its information to the output layer [9]. In this thesis, the output layer will predict the local CFL number.

Each layer consists of neurons, and each neuron in a layer is connected to all the neurons in the next layer, as shown in figure 3.2. In this figure we can see that each neuron in the input layer has a connection to all of the neurons in the hidden layer. Each of these connections has a certain weight w_i, which is shown on the connections between the hidden layer neurons and the output layer neuron. During training, these weights are adapted in order for the network to learn a certain task.

We can mathematically describe how information is passed from one layer to another. We will use figure 3.2 as the example network. From the input layer to the hidden layer we see six connections and each connection has its own weight. Furthermore a bias b is added to possibly shift the activation function. This activation function maps the output to a certain range and ensures that the learning algorithm will be able to make progress in each step [9].

The information from the input neurons of the network shown in figure 3.2 is passed through the neurons in the hidden layer in the following way [25]:

$$\sigma_1\left(w_{1,1}^{(1)}x_1 + w_{2,1}^{(1)}x_2 + b_1\right) = h_1^{(1)},$$
$$\sigma_1\left(w_{1,2}^{(1)}x_1 + w_{2,2}^{(1)}x_2 + b_2\right) = h_2^{(1)},$$
$$\sigma_1\left(w_{1,3}^{(1)}x_1 + w_{2,3}^{(1)}x_2 + b_3\right) = h_3^{(1)}.$$

Here, σ_1 is the activation function, $w_{i,j}^{(k)}$ are the weights of the connections between the input layer and the hidden layer in the kth layer and x_i are the values of each input neuron shown in figure 3.2. The network learns a task by adapting the weights and bias in each iteration by following the backpropagation algorithm.

The activation function that we use is ReLU, shown in figure 3.3, which is defined as

$$\text{ReLU}(x) = \max(0, x).$$

This function is not differentiable at $x = 0$, however this activation function works very well, is cheap and efficient, and does not show any problems when training the neural network [9].

In the last layer of the network, there is no activation function. The CFL function can be any real value except zero, since we will otherwise divide by zero in the pseudo time-step given in equation (3.5). Therefore, by omitting the activation function in the last layer, the network can output any real number.

3.4.2. Backpropagation

When a network is trained, each iteration the network obtains inputs and computes an output for every neuron in the network. With this forward pass, the network makes the predictions in the output layer and measures the error of this prediction with regards to the desired output. The function that computes this error is called to loss [9].

After the loss value has been computed, the network determines for each neuron in the last hidden layer its contribution to the error. Then, each neuron in the last hidden layer determines for the neurons in the previous hidden layer what their contribution to the error is, and so on, until the input layer. Gradient descent then measures the local gradient of the loss with respect to the predicted value of the network, and moves towards the direction of the descending gradient [9]. That means we have to compute the derivative of the loss function with respect to the learnable parameter in order to apply backpropagation. This derivative is propagated backwards into the neural network in order to train the network. This means that this procedure first determines how much each individual neuron in the last hidden layer contributed to the error, then it propagates backwards into the network to determine the contribution of each layer to the error, until the input layer is reached. Finally, the algorithm adapts the network weights to reduce the loss [9]. This whole process of propagating backwards and adapting the weights is called backpropagation.

In this thesis, adaptive moment estimation (Adam) will be used as a gradient descent method [9].

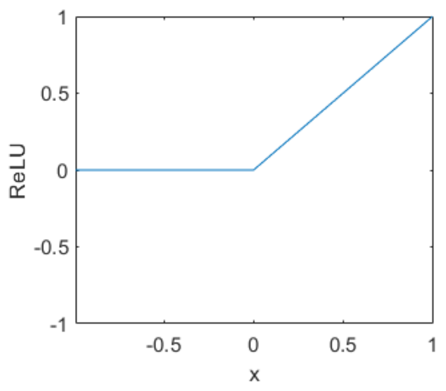


Figure 3.3: ReLU$(x) = \max(0, x)$ activation function.

3.4.3. Early stopping

Early stopping is a regularisation method that prevents the neural network to overfit. Overfitting means that the network can perfectly predict the target CFL number for the given training data, but fails to give an accurate prediction for data outside the training data set. Not only does early stopping prevent overfitting, it also speeds-up the training process of the network [9].

With early stopping, a parameter called “validation patience” is chosen. The validation patience is the number of times that the validation loss can be larger than the previously smallest loss before breaking out of the training loop. When the performance of the validation set is getting worse, the training of the network will be stopped when the validation patience has been reached. Thus, the network is regularised in terms of maximum number of trained epochs with a higher validation loss than the epoch with the smallest validation loss.

The validation patience cannot be too small, since the training is then stopped too early [9]. After the training has stopped because the maximum number of epochs have been reached or the validation patience has been reached, the network with the smallest validation loss is picked for testing on other kinds of laminar fluid flow models.

4 Methods

To predict the optimal CFL numbers for pseudo time-stepping, a neural network is created and trained. Two different loss functions for the neural network will be described. One loss is the root mean squared error (RMSE) between the network prediction and an optimised CFL number that serves as a target. The other loss does not use a typical loss function. Instead of using a training target, the loss and its derivatives are computed in COMSOL. In order to be able to do that, the network has an extra input layer and a concatenation layer.This extra input layer provides the network the coordinates of the predicted CFL number, to be able to put them back into COMSOL.

In this section, first the two loss functions are discussed. After that, determining the optimal CFL number in COMSOL is discussed. This optimised CFL number serves as a target for one of the loss functions, and provides insight how the optimised loss looks like. Then, the data sets to train the network is discussed. Finally, the networks with the two different loss functions are discussed. These two networks will be compared regarding the training and the network predictions of the input data, and the best network will be used for further research.

4.1. Network loss

As explained in section 3.4.2, a loss or cost function is defined to measure the error of the network prediction and the desired outcome. In this research, the network predicts local CFL numbers to improve the pseudo time-stepping method explained in section 3.3. The pseudo time-stepping method is improved if the predicted CFL numbers result in acceleration of convergence and if the neural network is able to generalise. The backpropagation algorithm in section 3.4.2 then adapts the weights of the network to minimise the the loss.

In this section, two possible loss functions are discussed. First, a classic root mean squared error (RMSE) loss is considered [9]. This loss uses a target for the network to train. Then, a second loss is discussed. This second loss does not use a target, but directly uses the network predictions to compute one pseudo time-step in COMSOL. An error estimate is then created to function as the loss.

4.1.1. Root mean squared error loss

The RMSE uses network targets for the network to learn. The targets in this case are optimised CFL numbers computed beforehand in COMSOL. The optimisation step in COMSOL is explained in appendix B and is further explained in section 4.2.2.

Let CFL_{pred} be the network prediction and CFL_{opti} be the optimised CFL number in COMSOL. The RMSE loss, like in [9], is then given as:

$$\text{loss} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\text{CFL}_{\text{pred}}^{(i)} - \text{CFL}_{\text{opti}}^{(i)}\right)^2}, \tag{4.1}$$

for the prediction of n CFL numbers. As we can see, when the network predictions CFL_{pred} are close to the targets CFL_{opti}, the loss value will decrease.

4.1.2. Error estimate loss and gradients

This loss is not a typical loss function. We want the network to predict local CFL numbers, such that the pseudo time-stepping in COMSOL is accelerated. So, minimising a function that estimates an error estimate of the simulation possibly results to better local CFL predictions and faster convergence. The error estimate that is considered for this loss is defined as:

$$\text{error estimate } = \left(\int_\Omega (u - u_{ex})^2 + (v - v_{ex})^2 \,\mathrm{d}\Omega\right)^{\frac{1}{2}}. \tag{4.2}$$

In this error estimate, u and v are the solutions after one pseudo time-step iteration computed with the predicted CFL number from the network. The solutions u_{ex} and v_{ex} are obtained by a converged solution computed in COMSOL. These solutions are thus rather numerical solutions than exact solutions. The integral is computed in COMSOL by defining an integral function.

This error estimate does not take into account that CFL numbers of zero are not allowed. Note in equation (3.5) that if the CFL number is zero, the pseudo time-step will be zero and thus the sequence will be divided by zero. So, in order to create a loss that penalises CFL numbers close to zero, an extra term is added. This term vanishes as the CFL number increases. The final loss function then becomes:

$$loss = \left(\int_\Omega (u - u_{ex})^2 + (v - v_{ex})^2 \,\mathrm{d}\Omega\right)^{\frac{1}{2}} + 1\text{E-}6 \cdot \sum \frac{1}{\text{CFL} + 10^{-8}}. \tag{4.3}$$

With this loss, CFL numbers near zero are penalised and the bigger the CFL number gets, the less influence this extra term has on the total loss. This factor especially plays a role at the beginning of training the network, when the outputs of the network are below one.

For training the network with a simulation in COMSOL of which we can obtain the converged solution, this loss function can be used. However, if we want to further train the network with other COMSOL simulations of which we cannot obtain a converged solution, another loss should be chosen. This could, for example, be a loss that minimises the residuals of the simulation.

Since this loss requires to compute a nonlinear iteration in COMSOL, the gradients also have to be computed in COMSOL. Normally, Matlab can compute the gradients of this network loss with the `dlfeval` function. However, since this is a very specific loss function that requires a computation step in COMSOL, the predictions of the network are not "traceable" anymore and Matlab is not able to compute the gradients of the loss itself.

In order to compute the network gradients, the derivative of the loss function with respect to the CFL number needs to be computed. Luckily, COMSOL can compute the derivative of the first term given in equation (4.2) and the derivative of the second term given in (4.3) is not difficult to determine.

The derivative of the first term is computed within COMSOL using a sensitivity analysis, explained in appendix B. The global objective, or the function of which we want to obtain the gradients, in this analysis will be the first term in the loss, given in equation (4.2). The network predictions are imported from Matlab via an interpolation function in COMSOL. This interpolation function obtains the network predictions and the corresponding coordinates and interpolates the discrete CFL points. The control variable of the sensitivity analysis will be the predicted CFL numbers, using the interpolation function with the predictions as an initial value. All what has to be done to compute this derivative, is updating the interpolation function data source, run the simulation for one step and obtain the results from the sensitivity analysis.

The derivative of the second part of the loss is easy to obtain by using the chain rule:

$$\frac{\partial}{\partial \text{CFL}} 10^{-6} \cdot \sum \frac{1}{\text{CFL} + 10^{-8}} = -10^{-6} \cdot \sum \frac{1}{(\text{CFL} + 10^{-8})^2}. \tag{4.4}$$

And we see that the closer CFL is to zero, the bigger the derivative in equation (4.4) becomes. Resulting that the network learns to stay away from CFL values near zero.

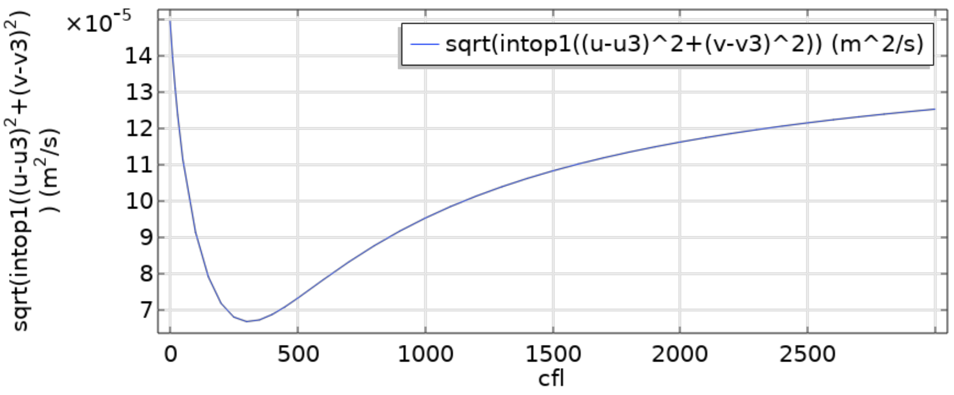


Figure 4.1: The outcome of the parametric sweep study for global CFL numbers ranging from 0 to 3000. The minimum of the error estimate is around a CFL number of 300.

So the derivative of (4.2) has a larger influence when the CFL number predictions are moving away from zero and (4.4) has a large influence in the beginning, when the predictions are near zero. The constant 10^{-6} is added so that the training progress will not be influenced later on in training, when the predictions are not near zero anymore. Because the predictions of the CFL number could be relatively small for certain elements, and this prediction should not be influenced by the term in (4.4).

4.2. Optimal CFL numbers

The neural network will be trained to predict the optimal CFL number. A local CFL number that enhances the pseudo time-step method by accelerating convergence is considered as an optimised CFL number. In order to know if the networks output optimal local CFL numbers and in order to compute targets for the error estimate loss in section 4.1.1, local CFL values are optimised in COMSOL using an optimisation study. This optimisation study is further explained in appendix B.

For all the CFL optimisation methods, a simulation in COMSOL of back-step (1) of table A.1 is considered. This laminar flow model is described in appendix A. The model has a maximum element size of 0.00186 m and an inflow velocity of 0.003 m/s.

First, a global optimum for the CFL number will be determined, by using a parametric sweep of different CFL numbers. In this case, the CFL numbers are thus the same over the whole domain. After that, optimisers in both COMSOL and Matlab will be used to compute local CFL numbers. In COMSOL, an optimisation study will be used, and in Matlab, the Adam optimiser is implemented to compute the optimal local CFL.

4.2.1. The optimal global CFL number in COMSOL

Finding the optimal global CFL number is done with a parametric sweep in COMSOL. The parametric sweep computes one pseudo time-step iteration with different CFL numbers. The CFL numbers go from 0 to 40 in steps of 10, 50 to 500 in steps of 50 and from 500 to 3000 in steps of 100. For each CFL, the value of (4.2) is computed.

The result of this parametric sweep is shown in figure 4.1. It is clear that there is one global optimum around a CFL number of 300, with a loss of $6.68 \cdot 10^{-5}$. In the next sections we will compare this result with the optimisation results for local CFL values. This optimal global CFL value will furthermore be used as an initial condition for the Adam optimiser and for the optimisation step in COMSOL in section 4.2.2.

4.2.2. The optimal local CFL number

There are two main reasons why it is important to obtain the local optimal CFL number: first, it gives an idea how well the network is able to predict the optimal CFL numbers, and second, these optimised CFL numbers are used as a target for the RMSE loss used for network training.

The function to minimise is the error estimate given in equation (4.2). Within COMSOL, the optimisation tool is used to find the optimal CFL number for a given step of the back-step simulation and

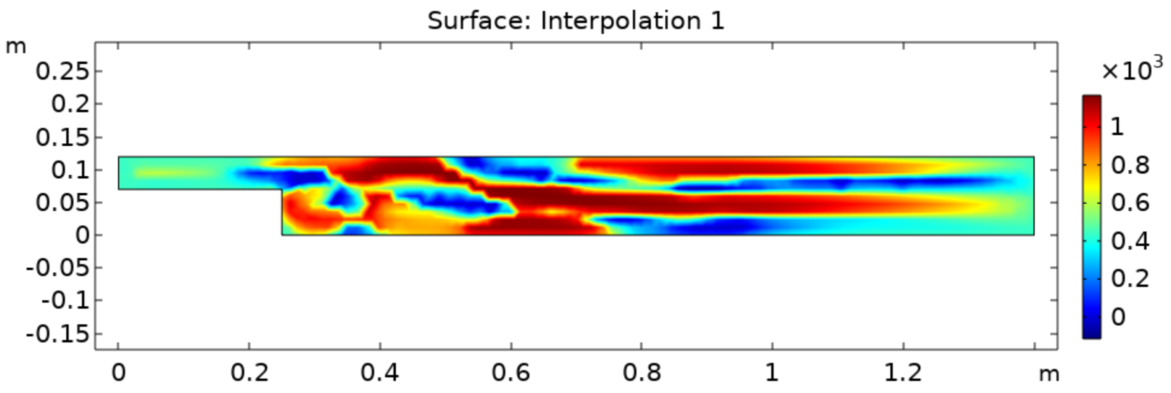


Figure 4.2: The CFL output of the Adam optimiser, where the loss was minimised to a value of $5.635 \cdot 10^{-4}$.

Adam is implemented within Matlab.

Adam optimiser in Matlab
Adam is short for *adaptive moment estimation* and in general determines the optimal value faster compared to the regular gradient descent algorithm [9]. In the standard gradient descent algorithm, the current value of the CFL number is updated with the use of learning rate α and the gradient:

$$\mathsf{CFL}_{n+1} = \mathsf{CFL}_n - \alpha \frac{\partial loss}{\partial \mathsf{CFL}}. \tag{4.5}$$

There are several improved versions of this method which are more likely to converge to the global minimum and with less iterations required, of which one of them is Adam. Adam is a combination of two of these optimisers, namely the momentum method and RMSprop [9].

In gradient descent with momentum, the current gradient is replaced by a momentum. The previous gradients play a role within this algorithm, where the momentum of this algorithm is computed as

$$\begin{aligned} m_{n+1} &= \beta_1 m_n - (1-\beta_1)\frac{\partial loss}{\partial \mathsf{CFL}}; \\ \mathsf{CFL}_{n+1} &= \mathsf{CFL}_n - \alpha m_{n+1}. \end{aligned}$$

The initial value for the momentum is zero. The parameter β_1 is the momentum friction [9]. The momentum uses the average of the past gradients to speed up the vanilla gradient descent method.

RMSprop stand for *Root Mean Squared propagation*, and the learning rate α is divided by a parameter v. This method is given as:

$$\begin{aligned} v_{n+1} &= \beta_2 v_n + (1-\beta_2)\cdot\left(\frac{\partial loss}{\partial \mathsf{CFL}}\right)^2; \\ \mathsf{CFL}_{n+1} &= \mathsf{CFL} - \frac{\alpha}{\sqrt{v_{n+1}+\epsilon}}\cdot\frac{\partial loss}{\partial \mathsf{CFL}}. \end{aligned}$$

Where the parameters β_2 and ϵ are respectively the decay rate and a very small number to prevent division by zero. The parameter v is used to speed up the algorithm when the gradients are small, by using the past squared gradients [9].

As mentioned earlier, Adam is a combination of gradient descent with momentum and RMSprop. That means it uses both the momentum m and the vector v. The CFL number update is computed as follows:

$$\mathsf{CFL}_{n+1} = \mathsf{CFL}_n - \frac{\alpha}{\sqrt{\hat{v}_{n+1}} + \epsilon}\cdot \hat{m}_{n+1}. \tag{4.6}$$

We see $\hat{v}$ and $\hat{m}$, which are the bias corrections of v and m computed as

$$\hat{m}_{n+1} = \frac{m_{n+1}}{1-\beta_1^{n+1}};$$
$$\hat{v}_{n+1} = \frac{v_{n+1}}{1-\beta_2^{n+1}}.$$

And again, the moment m and the vector v are computed the same as in gradient descent with momentum and RMSprop [9]:

$$m_{n+1} = \beta_1 m_n + (1-\beta_1)\cdot\frac{\partial loss}{\partial \text{CFL}};$$
$$v_{n+1} = \beta_2 v_n + (1-\beta_2)\cdot\left(\frac{\partial loss}{\partial \text{CFL}}\right)^2.$$

Adam was implemented in Matlab and applied on the back-step simulation in COMSOL to predict the optimal CFL number for the third step of the nonlinear solver. The third step was chosen, because the solution when using the nonlinear solver would have difficulties to result in a converged solution. Therefore, the optimised local CFL number could improve the solution for the fourth step in order to converge faster.

The Adam optimiser started with a learning rate α of 0.7, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and an initial CFL of 300. Due to the small gradients, a high initial learning rate was chosen and the initial is chosen to be around the optimal global CFL.

When the algorithm showed (heavy) oscillations during training or the loss would not decrease anymore, the CFL number which resulted in the smallest loss was picked as an initial condition and the learning rate was decreased. By doing so, the algorithm gradually converged to the minimum of the loss, which was $5.635 \cdot 10^{-4}$. The outcome of CFL values of the optimiser is shown in figure 4.2. The final learning rate used was 0.001, where the output of Adam would always result in the same CFL numbers and the same value for the loss.

Optimisation study in COMSOL

Within COMSOL, it is possible to optimise a value with an optimisation study, as explained in appendix B. This tool aims to find the minimum of a given global objective for a control variable field. The global objective in this case is the error estimate in equation (4.2) and the control variable field is the CFL number. That means COMSOL tries to find the CFL number that leads to the minimum of the loss.

In about twenty minutes the optimisation solver finished the computation and obtained a solution for this back-step. The optimisation can take more or less time, depending on, for example, the mesh size or Reynolds number. The CFL number obtained by the optimisation study is given in figure 4.3. With these local CFL numbers the loss was minimised to a value of $3.117 \cdot 10^{-5}$. This value could even be more minimised with different settings for the optimisation study, but it will cost significantly

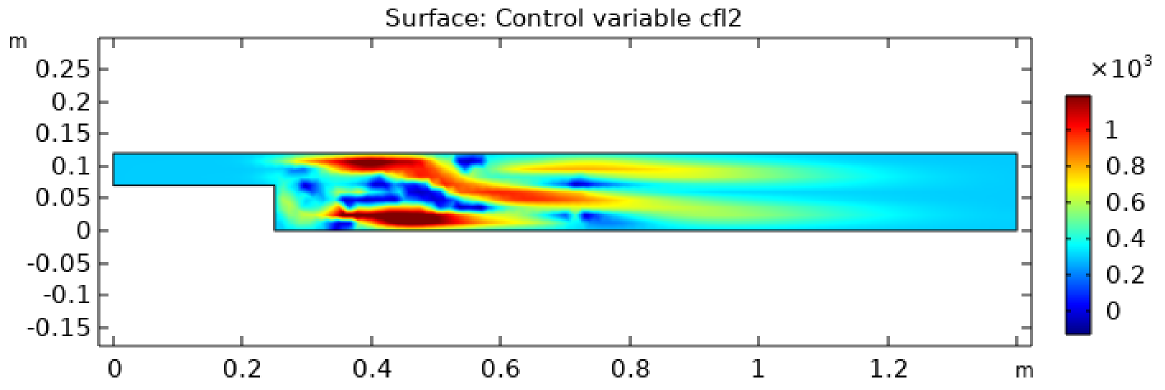


Figure 4.3: The CFL output of the optimisation study in COMSOL, where the loss was minimised to a value of $3.117 \cdot 10^{-5}$.

Variable	Unit	Location
Element edge length	m	-
u	m/s	Vertex
v	m/s	Vertex
p	Pa	Vertex
residual(u)		Vertex
residual(v)		Vertex
residual(p)		Vertex
spf.res_u	N/m^3	Vertex
spf.res_v	N/m^3	Vertex
spf.res_p	kg/(m^3· s)	Vertex
Cell Reynolds number		Centroid

Table 4.1: Variables used for input for the neural network with their given unit and location on the element.

more time. Right now, this result is sufficient to get an idea how the optimal local CFL number would approximately look like, to compare with the result of the Adam optimisation and to use as a target for the neural network.

When we compare the result of Adam in figure 4.2 and the result of the COMSOL optimiser in figure 4.3, we can clearly see that the structures are the same. They both show similarities in the negative areas as well as the areas where the local CFL is relatively high. That probably means the Adam optimiser probably has not converged yet. At the same time, the COMSOL optimiser takes less time to obtain an accurate solution.

Comparing the results of the local CFL numbers with the optimal global CFL number, we see that the local CFL number obtained by the COMSOL optimiser result in the smallest loss. The local solution of the COMSOL optimiser will be used to train a neural network, with this optimal local CFL number as a target.

4.3. Data sets

For the training of the networks, two different types of data sets will be considered. The data sets contain local information of a Newton step of a COMSOL laminar flow simulation. The first type of data points considered, are data points that only contain local information of one element, shown in figure 4.4a. The other type of data points contain information of a patch of elements, shown in figure 4.4b.

Here, local information is information obtained from element cells, element vertices and element edges. A patch of elements contains information of a central element and the three adjacent elements. Patches of elements are also used in [17, 18, 22], which seems to be an effective method to improve network predictions. Local information is used to achieve a generalisability. This means that the neural network is able to predict CFL numbers such that it accelerates convergence for laminar flow models on which it has not been trained.

In this chapter, the input data will be discussed, the construction of a single element as a data point and a patch as a data point will be discussed, and the composition of data obtained from different COMSOL laminar flow models is discussed.

4.3.1. Input data

Local information obtained from the mesh vertices will be used to create data to train a neural network. The choice of using local information for the network inputs has several benefits. Firstly, it is shown in [17, 18, 22] that it improves generalisability of the network. The unique model specific information is not used, but only information that is locally available for all laminar flow models is used. This results in a trained neural network that is applicable for different kinds of laminar flow models. Secondly, only one flow model can generate enough data points for training the network, as one single flow model already contains hundreds of mesh elements [17]. Although enough data points can be obtained from one flow simulation, other simulations are also considered to create variety of data within the whole data set.

An estimate has to be made which local data is relevant for predicting the optimal CFL number for pseudo time-stepping. In section 3.3, we see that for the computation of the pseudo time-step the velocity solution vector $\mathbf{u}$ and the element size are used. Therefore, information about the velocities u, v and the element size are natural choices as input data for the neural network. Instead of the element size, the element edge lengths are used as an input, as this gives extra information about the distance between the vertices.

Furthermore, information of the solution of the pressure p is added, as well as the two available residuals of u, v and p, and the cell Reynolds number Re. In Table 4.1, all the input variables are listed alongside their unit and the location of the information in the element. Values of the residuals give information about the local convergence of the COMSOL simulation. The residual `residual(u)` is the residual when the solution is filled in into the discritisation and `spf.res_u` is the residual obtained when the solution is filled in into the partial differential equation.

The Reynolds number in equation (3.2) provides information about the flow, and is also used in papers [17, 18, 23] as input for the machine learning model. The input data is also normalised, using the function `normalize` in Matlab [21]. The normalised data has centre 0 and a standard deviation of 1. The centre “C” and scale “S” are saved and will be applied to normalise data from another COMSOL simulation.

In order to achieve generalisability, not only using local data is important, but also the quality of the local data. It is reasonable to assume that a neural network struggles with giving correct predictions if the input values are not in the range of the training data. For example, if the input velocities are higher than the velocities that the network faced during training, it is to be expected that the network will not be able to give accurate predictions. So, in order to create a neural network that can make correct predictions for several kinds of laminar fluid flow simulations, the training data set needs to contain data points that vary in the values given in table 4.1.

What therefore is also important, is that the data set for network training should consist of information of simulations with different velocities, mesh sizes and Reynolds numbers. In table 4.2, data sets created from different back-step geometries with varying maximum element size and inflow velocities are shown. The neural network will be trained on a combination of all the data sets created from the four back-steps.

A single element as data point

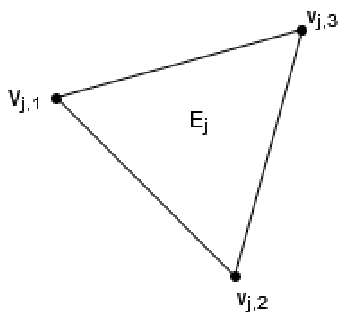
Introduce a triangular mesh element schematically shown in Figure 4.4a. This element has centroid E_j and three vertices $V_{j,1}$, $V_{j,2}$ and $V_{j,3}$. In Table 4.1, the variables located on the vertices $V_{j,1}$, $V_{j,2}$ and $V_{j,3}$, and the variables located on the centroid E_i are given. In total there are 31 inputs when a network is trained with this data set.

Using only the information of one element in the data set means that there will not be any information given of the surrounding elements for training. Although this is a simple method to generate data and training probably takes less time compared to a patch, the data points might lack information to properly train the neural network. A data point generated from information of one element requires a network with 31 neurons in the input layer.

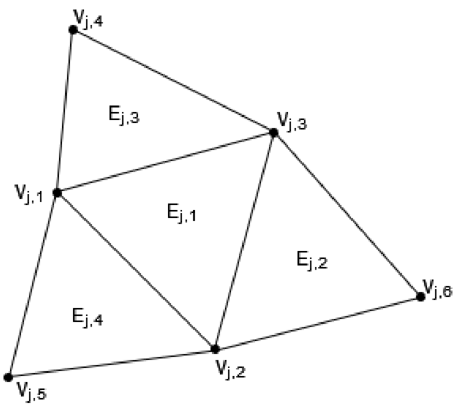
A patch as a data point

Using local information may be beneficial for the generalisability, but information that is too local can have the disadvantage that it might not contain enough information for the network to be able to learn the optimal CFL. That is why a data set with data points that contains information of a patch of elements will also be created and its effectiveness will be investigated. The patch consists of a central element and its four adjacent elements, as can be seen in Figure 4.4b.

Information of all the vertices of the elements in the patch will be in a data point. The data point order will first contain all the information of the central mesh element $E_{j,1}$, then all the information of element $E_{j,2}$ and so on to element $E_{j,4}$. The information of each element will be the same as the element information in section 4.3.1. This means the information on the vertices $V_{j,1}$ to $V_{j,3}$ are in the data point three times. Although this double information seems redundant, it can also add necessary information about how the elements are interconnected. Also, since it is not clear how COMSOL orders



(a) Data point which contains only information of one element E_j with vertices $V_{j,1}$, $V_{j,2}$ and $V_{j,3}$



(b) 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 $V_{j,6}$

Figure 4.4: Schematic figure of the two different data points.

the elements, it is difficult to make an ordering of the data points. This makes removing the redundant information also more difficult. Possible improvements on the input data is discussed in section 6.1.

The inner elements of the mesh all have three neighboring elements, but the elements facing the boundary only got two adjacent elements. Initially, the boundary elements were left out in the data set. However, after training and testing networks, it turned out that especially the information of the boundary elements can be important. Therefore, information of the boundary elements are included in the data set. One data point of a boundary element contains information of the element itself and the two neighboring elements. The “third” element “outside the domain” contains information of the two vertices on the boundary, including the edge length, but the information outside the domain will simply put to zero. The first test of this method to include boundary elements did not show any difficulties with the zero valued information and seemed to interpret it as boundary elements. Boundary elements with only one adjacent element were still left out, and for the meshes of the simulations considered in this project it was made sure that these elements were not in the boundary. Data sets created with element patches contain $4 \cdot 31 = 124$ inputs.

4.3.2. Composition of data

To create a data range sufficiently large to obtain generalisability, different simulations of a back-step model are created. The composition of data obtained from these COMSOL simulations is important. If the majority of data points consists information of only small elements, the network probably struggles with training the data points with information of larger elements. There has to be a balance between the data points within the data set.

Furthermore, in order for the network to be able to make good predictions for different simulations, the data in the data sets needs to contain as much different values of variables in table 4.1 as possible. Thus, different velocities, element sizes and cell Reynolds numbers should be represented in the data set, preferably evenly. In this section, the composition of the data sets obtained from different back-steps for training networks is explained.

To create data with a wide range of information, different back-step simulations are considered. In total there are four different back-steps. Two back-steps have different geometry, and the other two are scaled versions of the first two, with the same Reynolds number. The geometries for each back-step is given in Table A.1 in appendix A.

The different back-step simulations for each of the four back-steps are given in Table 4.2. For each of these simulations, different combinations of mesh size and inflow velocity are used to create a wide range of information. By doing so, the network is trained on a variety of different velocities, mesh sizes, and thus will be likelier to perform well on a wider range of different laminar fluid flow problems. These problems should probably have mesh size and velocities within the range of the training’s data, otherwise the network hasn’t learned these inputs and can probably not give a good prediction for the

Simulation	Number of elements	Max. element size (m)	Inflow velocity (m/s)	Iterations
(1) Back-step	6516	0.0156	0.001 0.005 0.01	1, 10 10 10
	3908	0.0206	0.003 0.006 0.009	1, 10 10 10
	2372	0.0266	0.002 0.007 0.015	1, 10 10 10
	1186	0.0266	0.008	2, 4
	13828	0.0106	0.004 0.008	2, 10 2, 10
(2) Back-step scaling of (1)	6516	0.00156	0.01 0.05 0.1	1, 10 10 10
	3908	0.00206	0.03 0.06 0.09	1, 10 10 10
	2372	0.00266	0.02 0.07 0.15	1, 10 10 10
	1186	0.00266	0.08	2, 4
	13828	0.00106	0.04 0.08	2, 10 2, 10
(3) Back-step	5808	0.0156	0.005	1, 10
	4134	0.0186	0.01	2, 10
	8790	0.0126	0.013	3, 10
(4) Back-step scaling of (3)	5808	0.00156	0.05	1, 10
	4134	0.00186	0.1	2, 10
	8790	0.00126	0.13	3, 10

Table 4.2: Different combinations of mesh sizes and flow velocities of the back-step simulations to create 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 the appendix table A.1.

CFL number.

In order to create a wider range of different cell Reynolds numbers for the input, two back-steps have different proportions than the other two. With these proportions, back-steps (1) and (2) in Table 4.2 have relatively low cell Reynolds numbers, and back-steps (3) and (4) have cell Reynolds numbers with a higher order.

For all the data points, the optimal CFL number per element is computed as in section 4.2.2. This CFL number is used as a target for one of the loss functions.

In all data sets, the data points containing outliers of the CFL numbers are removed to improve network training. For some of the obtained data points, the absolute value of the CFL number was significantly larger than the other CFL numbers in the data set. Including these outliers can disturb the training process of the network. The network is trained to minimise the loss, and the data points that include an outlier can lead to high loss value. So instead of learning the optimal CFL number per data point, the network is more likely to take the mean of outliers and the other values for the CFL number. Therefore, the CFL values that are (much) larger in absolute value compared to most CFL numbers in the data set are removed.

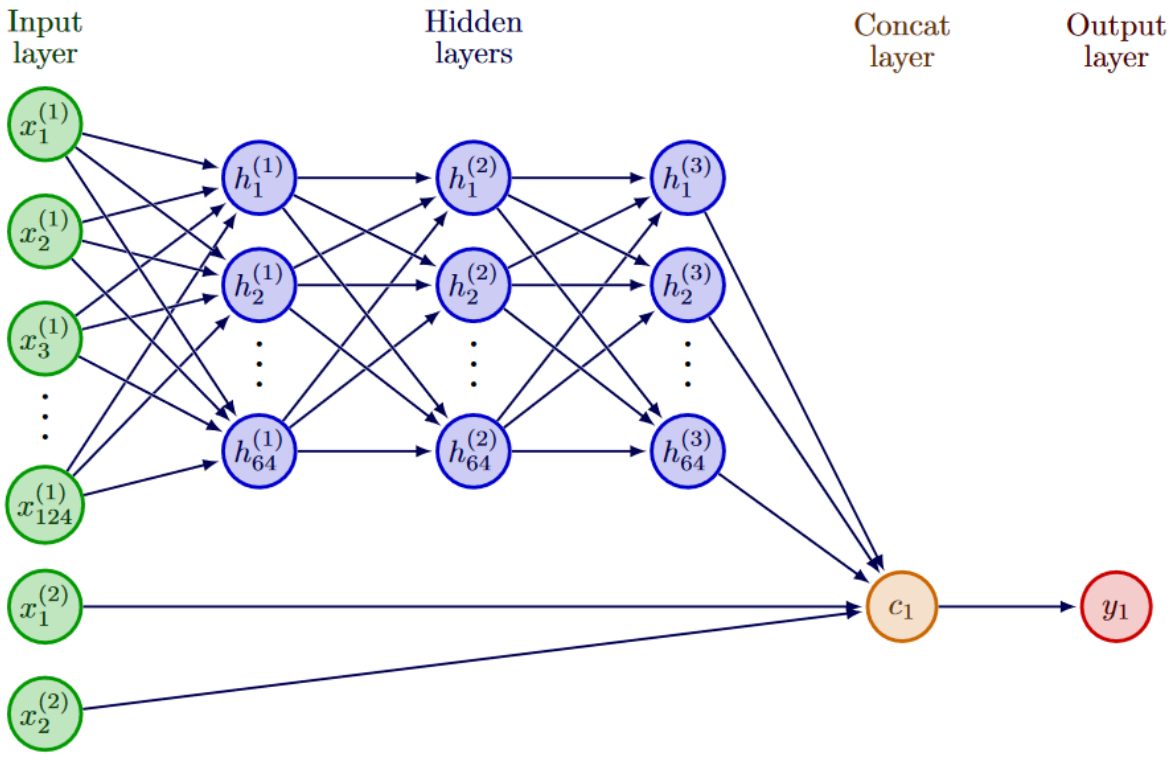


Figure 4.5: Neural network with the two input layers $x^{(1)}$ and $x^{(2)}$, 3 hidden layers with 64 neurons, a concatenation layer that connects output of the hidden layers with the second input layer, and finally the custom regression layer or output layer y_1.

4.4. Neural networks to predict the optimal CFL number

In this research, two different losses are considered. Each loss requires its own network structure. First, the network which uses the error estimate loss in (4.3) is discussed. Then, the network that uses the RMSE loss in (4.1) will be discussed. Finally, the training and preliminary results on a back-step is discussed.

4.4.1. Network with error estimate loss

When using the error estimate loss, a special structure for the network is required. Since the network predictions have to be put back into COMSOL each training iteration, a special layer in Matlab has to be created in order to compute the loss and its gradients. Furthermore, in order to be able to put the local CFL numbers back in into COMSOL, the network must also know the corresponding coordinates of the CFL predictions. For this purpose, an extra input layer is created to provide necessary information for the network to compute the loss and gradients in COMSOL.

The structure of this network is schematically shown in Figure 4.5, where we see the two input layers $x^{(1)}$ and $x^{(2)}$ are also shown. Depending if patched data or single element data is used, input layer $x^{(1)}$ has either 31 or 124 neurons to predict local CFL numbers. After this layer there are several hidden layers that predict the CFL number. The second input layer $x^{(2)}$ always has two neurons: one for the x coordinate and one for the y coordinate of the CFL prediction.

With the concatenation layer, the CFL prediction and the coordinates are concatenated and forwarded to the output layer. This output layer is a so-called "custom regression layer", which can determine the loss (4.3) and its gradients in COMSOL using the CFL prediction and coordinates. Since the output of the neural network is used as an input in COMSOL to compute the error estimate, the loss itself is not "traceable" anymore in Matlab. That means that Matlab is not able to compute the gradients based on the custom loss function. .

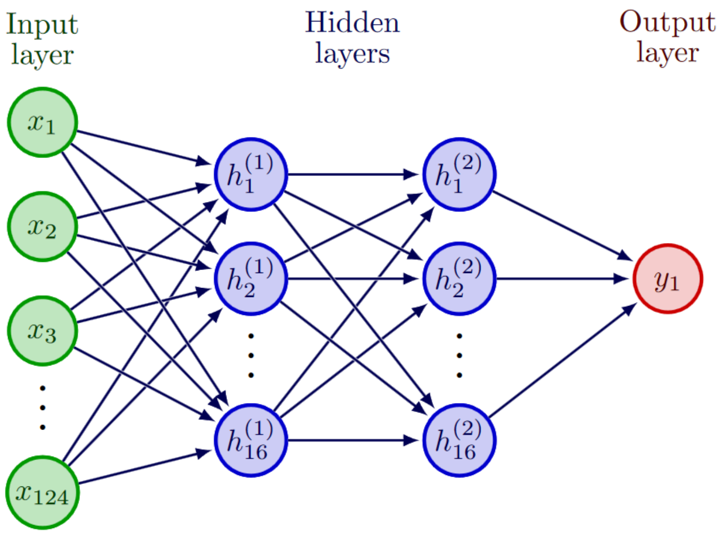


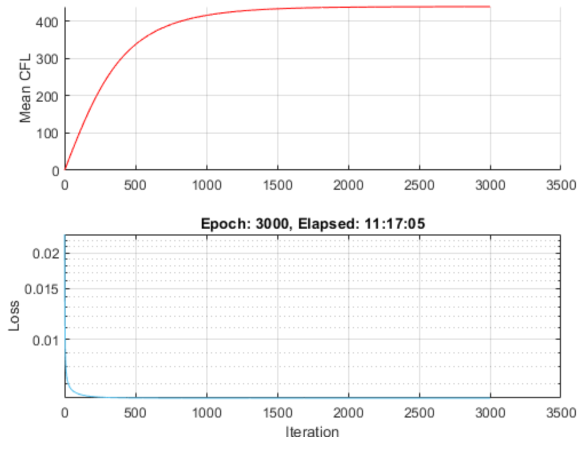
Figure 4.6: Neural network with 124 inputs, 2 hidden layers with 16 neurons and an output layer.

4.4.2. Network with RMSE loss

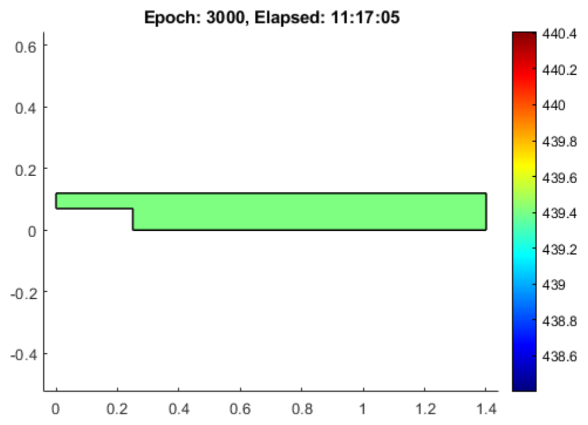
Using the RMSE loss in (4.1) to train a network, the network will have a standard structure. It has one input layer with either 32 or 124 neurons, depending if the data set used information of a patch of neurons or information of a single element. Then, there are several hidden layers, with a number of neurons. The optimal amount of hidden layers and number of neurons per hidden layer have to be determined by a grid search. In [22], the network had 5 hidden layers with each 20 neurons, so it is expected that the network has to be very large in order to learn the target. In section 5.1 the grid search is performed. After the hidden layers, there is one output layer that predicts the optimal CFL numbers.

4.4.3. Network training

For both loss functions a network will be created and trained in order to check if it is able to predict the optimal CFL number. The predictions of each network will be compared to to the optimised CFL number, computed in section 4.2.2. Furthermore, the training time and loss value of each network will be considered to determine how well each type of network is able to predict optimal CFL numbers for pseudo time-stepping.

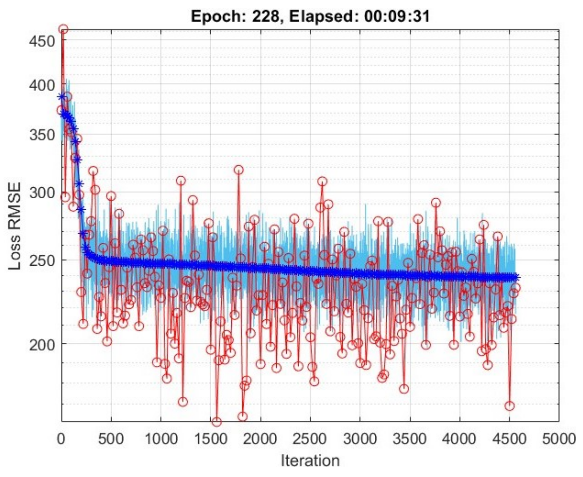


(a) Training of the network, with the mean CFL prediction in red and the loss in blue

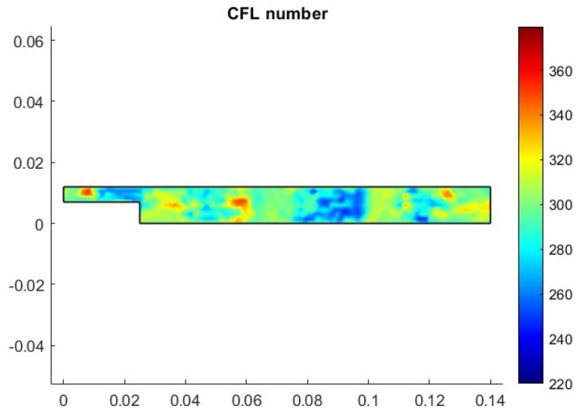


(b) Predicted CFL number of the last epoch during training.

Figure 4.7: The training plot of network the network, with objective to minimise the solution error, shown on the left, with the corresponding predicted CFL on the right.



(a) Training of the network, with mean loss over an epoch in dark blue, loss per iteration in light blue and the validation loss over an epoch in red.



(b) Prediction of the network for back-step (2) in table A.1 with inflow velocity 0.07 m/s and maximum element size 0.00186.

Figure 4.8: The training plot of network trained on the optimised CFL targets shown on the left, with the corresponding predicted CFL on the right.

Network with error estimate loss

For the network described in section 4.4.1, we use data from one Newton step. Since this network computes the loss directly from a COMSOL simulation, all the data points from one Newton iteration has to be used during one step. It is also possible to create mini-batches, so sequentially train the network with subsets of the training set [9]. However, the CFL number on the remaining elements outside the mini-batch has to be filled in in some way.

There are several options to do that. Firstly, it is possible to fill up the remaining CFL numbers with the predicted global CFL in section 4.2.1. Secondly, predictions from previous steps could be used together with the prediction of the current step in order to compute the loss. With this option only the initial value of the CFL number has to be determined.

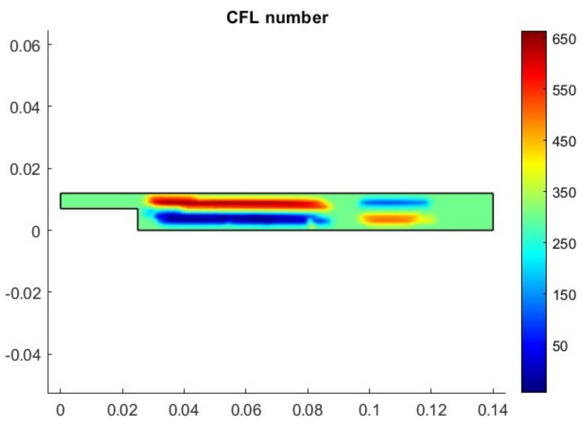
Unfortunately, none of these options result in a network that is able to minimise the loss within a reasonable learning time. This network has been applied to back-step (1) in table 4.2 with maximum element size 0.00186 m and an inflow velocity of 0.003 m/s. As we can see in Figure 4.7a, the training takes more than 11 hours, and in Figure 4.7b we can also see that the predictions are the same for the whole domain after this long time of training. The loss only decreases significantly in the beginning of the training, probably due to the second term in (4.3). One could think that the network first learns to train the global optimum for the CFL number, which is around 300 to 400 as we can see in Figure 4.1, and that the network will then try to take different values for each element. That was unfortunately not the case. The network will output the same CFL number for each element, namely 439, after more than 11 hours of training. When the network weights are checked after training, they are all around zero.

Another drawback of computing the loss and gradients within COMSOL, is the fact that for each iteration in the training, the network has to call and compute one nonlinear iteration in the COMSOL simulation. As we can again see in Figure 4.7, 3000 epochs or 3000 iterations already costs more than 11 hours of training time.

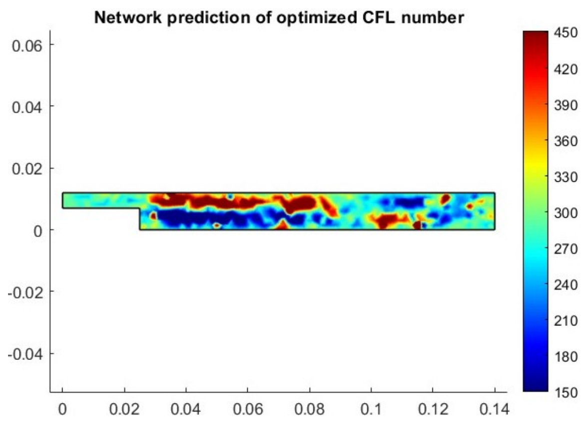
Network with RMSE loss

For the second network, described in section 4.4.2, data from four different back-steps is used to train the network. Mini batches are used, and both the training data and validation data are divided into 10 mini batches. So for each iteration, 10% of the training data is used for the network training. After 10 iterations one epoch has passed and the training data will shuffle again into 10 randomly assigned mini-batches [9]. The same holds for the validation data, except that 10% of the validation data is used for each epoch and will be shuffled randomly after 10 epochs.

In Figure 4.8a we see that the mean loss (in dark blue) decreases fast in the beginning of the training and that the validation (in red) follows that trend. In order to obtain a network that is not overfitted to



(a) Optimised CFL target of back-step (2).



(b) Prediction of network trained on back-step (2), which was very overfitting.

Figure 4.9: The optimised CFL target and overfitted network prediction. This shows that the neural network is able to learn the CFL target with the given input data.

the input data, early stopping is used. The network which resulted in the smallest validation will be the obtained network after training. The network stops the training if the validation patience of 150 has been reached. If within these 150 epochs has not been a validation loss less than the first validation loss, the training is aborted.

In Figure 4.8b we see that the prediction of the network is not the same over the whole domain. Since this network is trained on a combined data set over back-steps (1) to (4), described in table 4.2, the prediction of the network in this case does not look like the optimised CFL number by the COMSOL optimisation solver.

If the network was only trained on data generated by back-step (2) and applied to back-step (2), the prediction would look like the optimised CFL target. This is shown in figure 4.9. In 4.9a we see the CFL target for one of the data sets created from back-step (2) and in 4.9b we see the prediction of the network. Note that this network is overfitted and the validation loss was much higher than the training loss. This example is only used to show that the prediction of the network has the same silhouette as the target. The maximum and minimum of the colour scale are adapted for this prediction, to make the red and blue part more visible. This figure shows that the neural network is able to learn the optimised CFL target with the given input data. Therefore, we conclude that it is possible to train a network with these input data to predict the optimised local CFL number for pseudo time-stepping. And thus, for this research, this network will be used for CFL prediction.

5 Trained networks and results

With a data set containing 48000 data points created by the four different back-steps shown in table 4.2 and a network trained on an optimised CFL target, the network's CFL prediction will now be compared to the two other CFL number in COMSOL, given in equations (3.7) and (3.8). In order to compare the effectiveness of each CFL number, the required number of nonlinear iterations are compared in this section. Since the network predictions are made in Matlab and the interface between COMSOL and Matlab, using the number of nonlinear iterations is chosen to measure how fast the network predictions lead to convergence.

Before the neural network will be applied to different kinds of laminar flow simulations in COMSOL, a grid search has to determine which structure of hidden layers and neurons result in the network that can generalise and predict the CFL numbers best.

After the grid search, the importance of using a combination of data sets from different COMSOL simulations is illustrated. Lastly, the network will be applied to different kinds of back-steps, Couette flows and cavity flows. All these COMSOL simulations are explained in appendix A. For one back-step simulation and one Couette flow an obstacle will be placed within the geometry to test how well the network deals with those simulations. Furthermore, in appendix C, results of either using element patches or single elements as data points are given. Here, the data patches performed better than data of single elements.

For all the simulations ran with the network's CFL number, the maximum number of nonlinear iterations is set to 100, and for the COMSOL CFL it is set to 150. That means that the computation is automatically aborted if this number of nonlinear iterations has been reached, regardless if the solution has converged.

The convergence of the simulations is determined by an error estimate, given by global variable probe in COMSOL:

```
sqrt(intop1(residual(u)^2+residual(v)^2+residual(p)^2)).
```

If this error estimate is below a certain value, the solution has converged. If both COMSOL CFL numbers result in convergence, the largest value of both error estimates will be used to determine when the simulation is run with the network's CFL number. In the case that only one converges within 150 iterations, the error estimate of the converged solution will be used and if both COMSOL CFL numbers do not converge, the limit for convergence is put to 10^{-7}.

Furthermore we will refer to CFL$_{iter}$ for the CFL number computed by the nonlinear iteration number given in equation (3.7), and to CFL$_e$ for the CFL number computed by the nonlinear error estimate given in equation (3.8). The results obtained from the network's CFL number will be compared to the results when CFL$_{iter}$ and CFL$_e$ are used.

5.1. Hyper parameter optimisation

In [22], they used network has 5 hidden layers with each 20 neurons. The other two papers [17, 18] unfortunately do not mention the used network parameters. To get an idea how large the network

Layers	Neurons	Test	Train	Validation
5	256	260.842	234.4	172.54
4	256	261.8668	243.2	170.24
5	64	269.3783	266.89	183.43
3	256	270.3051	258.24	179.15
3	128	270.3895	267.54	174.65
4	64	271.0954	270.7	178.46
4	128	271.2943	259.02	175.38
5	128	280.4108	256.22	183.07
3	32	280.8498	287.3	185.46
4	32	283.4153	284.5	187.89
3	64	283.9667	279.11	188.59
4	16	300.256	307.96	201.02
2	64	302.6808	297.35	204.32
2	32	306.6738	317.18	200.65
3	16	308.2663	319.22	204.59
2	16	313.7875	324.28	205.84

Table 5.1: Grid search of networks with different hidden layers and neurons per hidden layer, including the mean loss of 6 networks over the test, validation and train data set. From smallest to largest test loss.

should be to generalise well and accelerate convergence, a grid search is performed. Two different methods are considered.

First, different networks with varying number of hidden layers and neurons per hidden layer are created. Per combination of hidden layers and neurons, 6 networks are trained. A test set is applied to each of the trained networks, and the mean value of the loss over the 6 networks with the same size is taken. The results of the loss over the test set will be compared to each hyper parameter combination.

Secondly, since the loss does not necessarily give an impression how well the network can actually perform on other models besides the back-step, a small, a large and a network with a size in between will be trained. These networks will be applied to a both a back-step and a Couette cylinder flow simulation in COMSOL. The geometry and these simulations are given in appendix A.

The grid search is performed with networks with 2 to 5 hidden layers, and 16 to 256 neurons per hidden layers. For each combination of number of hidden layers and neurons per layer, 6 networks are trained and the loss over the test, validation and training set are computed.

In table 5.1, the results for the grid search, including the mean value of the loss over the test, validation and train data set of the 6 networks, are given. Recall that the loss of this network is the RMSE of the optimised CFL number and the network output. As we can see in this table, the networks with 256 neurons and 4 or 5 hidden layers have the smallest loss over the test set. Networks with the largest loss over the training set are the much smaller networks, with 2 or 3 hidden layers and 16 to 64 neurons per layer.

Since the network loss is based on the optimised CFL number and not on the residual when the predicted CFL is used, it is hard to conclude form table 5.1 which of the networks performs best in terms of generalisability and required number of nonlinear iterations per COMSOL simulation. Furthermore, in [22], the network was much smaller. Therefore, an additional test is performed to check how a large network, a small network or a network with a size in between the two perform when applied to COMSOL simulations.

The three networks are trained and applied to both the back-step and a Couette cylinder flow. The dimensions of the three networks are given in table 5.2. Network 1 has 4 hidden layers and 256 neu-

	Network 1	Network 2	Network 3
Hidden layers	4	3	2
Neurons	256	64	16

Table 5.2: Network dimension of network 1, 2 and 3 in table 5.3.

Grid search, back-step (1)								
	CFL$_{iter}$	CFL$_e$	Network 1	Network 2	Network 3	Equal	None	Total
Best	0	0	1	1	7	12	3	**24**
No convergence	6	2	8	8	4	-	-	**-**
Grid search, Couette								
	CFL$_{iter}$	CFL$_e$	Network 1	Network 2	Network 3	Equal	None	Total
Best	0	0	13	1	5	11	0	**30**
No convergence	1	1	0	27	2	-	-	-

Table 5.3: Summary of the grid search performed on back-step (1) in table D.1 and on the Couette flow in table D.2. For each CFL number, the amount of simulations on which it performed best is given. That is, the CFL number that required the least amount of iterations to obtain a converged solution. Also, the number of simulations that did not converge with the given CFL number are shown. The column “Equal” gives the amount of simulations for which two or more networks performed equally well.

rons per hidden layer, network 2 has 3 hidden layers and 64 neurons per hidden layer and network 3 has 2 hidden layers and 16 neurons per hidden layer.

In table 5.3 the summery of the results of the three networks are given. In case of the back-step, network 3 performs better than the other two networks for 7 simulations. Network 1 and 2 were only the best option is one of the simulations. Network 3 is also able to lead to a converged solution for more simulations, compared to network 1 and 2. Still, for halve of the simulations the networks perform equally well.

It is clear that network 3 performs best for the back-step, but since we are also interested generalisability, the three networks are also applied to a Couette flow. This Couette flow has an outer cylinder of 0.4 m, an inner cylinder of 0.2 m, and the inner cylinder is rotating with a certain wall velocity. The summary of the results of this analysis can also be found in table 5.3.

In this table, we can see that network 1 performs best for 13 simulations. In none of the cases, the CFL numbers defined in COMSOL are the better option. Network 3 is the best option in 5 of the 30 simulations. Network 2 does not lead to convergence in a lot of cases, thus this network will be eliminated.

From table 5.3 network 1 with 4 layers and 256 neurons, and network 2 with 2 layers and 16 neurons are the two networks that show the best performance. In table 5.1, network 1 scores high in the grid search, whilst network 3 scores the lowest with the highest test loss.

For the back-step, network 3 turns out to be the best network. Also network 1 performs very well, as in most cases the number of required nonlinear iterations does not deviate far from the required number of iterations for network 3. There are some cases where network 1 (and network 2) do not lead to convergence, while the predictions of network 3 do lead to a converged solution. Therefore, from these results we conclude network 3 is the best option.

Still, since generalisability of the network is also important, the results from the Couette flow should definitely be considered before making a decision on the network structure. As we see in table 5.3, both the predictions of network 1 and 3 lead to convergence in most of the cases. In most cases, network 1 outperforms network 3. Only in 5 of the 30 cases network 3 is the best option, compared to 14 cases for network 1. So, only taking these results into account, network 1 is definitely the best option.

However, network size is also something that needs to be taken into account. Shallower networks are less prone to overfitting than deeper networks, and have shown effectiveness in [22]. Since network 3 is much smaller than network 1, but still is performing better or equally well in a lot of cases for the Couette flow and in more cases for the back-step compared to network 1, network 3 is chosen to be used for generating results further on. So, the network with 2 hidden layers and 16 neurons will be used from now on.

	Combined	DS1	DS2
Trained on	Back-steps (1) to (4)	Back-step (1)	Back-step (2)

Table 5.4: Network names and training sets for networks in table 5.5.

Back-step (1)								
	CFL$_{iter}$	CFL$_e$	Combined	DS 1	DS 2	Equal	None	Total
Best	0	1	15	1	0	1	2	**24**
No Convergence	6	3	4	6	7	-	-	-
Back-step (2)								
	CFL$_{iter}$	CFL$_e$	Combined	DS 1	DS 2	Equal	None	Total
Best	3	0	4	2	1	10	4	**24**
No Convergence	6	4	6	8	8	-	-	-

Table 5.5: Summary of results of back-steps (1) and (2) in tables D.3 and D.4 with networks in table 5.4. For each CFL number, the amount of simulations on which it performed best is given. That is, the CFL number that required the least amount of iterations to obtain a converged solution. Also, the number of simulations that did not converge with the given CFL number are shown. did not converge with the given CFL number are shown. The column “Equal” gives the amount of simulations for which two or more networks performed equally well.

5.2. Results network trained on combined data set

From now on we will only consider a network with 2 hidden layers and 16 neurons per hidden layer, trained on the data set which contains information of all the 4 different back-steps shown in table 4.1. From each of the four back-steps, 12000 data points are selected randomly, resulting in one large data set that contains 48000 data points.

This network will first be applied to these four back-steps on which it has been trained on. Furthermore, to show the effectiveness of training on the combined data of back-step (1) to (4), two neural networks that has been respectively trained on back-step (1) and back-step (2) will also be considered. These two networks will be applied to back-step (1) and back-step (2). It is expected that the network trained on back-step (1) performs well on back-step (1) and not on back-step (2), and vice versa.

Furthermore, the network trained on back-steps (1) to (4) will also be applied to back-step (1), but with a rotated and mirrored geometry. This in order to show that the network is able to generalise on different geometries of back-steps.

After the network has been applied to different back-steps, we will apply the network to different types of laminar flow simulations in COMSOL. A Couette cylinder flow, a cavity flow, a back-step with obstacle and a Couette flow with obstacle will be considered to show that the network is able to generalise and thus accelerate convergence for these laminar flow simulations.

5.2.1. Results on back-step

In order to see the benefit of combining data sets that contain information of different back-step simulations, we will compare the results of networks trained on data from back-step (1) and back-step (2) of 4.2, and of the network that is trained on all data sets of 4.2. It is expected that the network on the combined data set is likely to converge within the velocity and mesh size range of the combined data set. Furthermore, the network trained on back-step (1) will probably not perform as good on back-step (2) as the network trained on back-step (2), and vice versa.

After these three different networks have been compared, the network trained on the combined data set will also be applied to back-steps with the geometry of back-step (3) and back-step (4) of table 4.2, as well as a rotated and mirrored version of back-step (1), in order to show that the network can generalise on different geometries of back-steps.

Back-steps from data set

For the back-steps in table 4.2, different combinations of mesh sizes and inflow velocities are created. For each simulation, the neural networks have been applied. The different mesh sizes and inflow velocities will be within the range of the training data, shown in table 4.2.

Back-step (1)						
	CFL_{iter}	CFL_e	Network	Equal	None	Total
Best	0	2	15	5	2	**24**
No Convergence	6	3	4	-	-	-

Back-step (2)						
	CFL_{iter}	CFL_e	Network	Equal	None	Total
Best	5	0	3	13	3	**24**
No Convergence	6	4	6	-	-	-

Back-step (3)						
	CFL_{iter}	CFL_e	Network	Equal	None	Total
Best	0	0	15	0	1	**16**
No Convergence	16	6	1	-	-	-

Back-step (4)						
	CFL_{iter}	CFL_e	Network	Equal	None	Total
Best	1	9	1	0	5	**16**
No Convergence	15	6	15	-	-	-

Table 5.6: Summary of the results on back-steps (1) to (4) in tables D.3, D.4, D.5 and D.6. For each CFL number, the amount of simulations on which it performed best is given. That is, the CFL number that required the least amount of iterations to obtain a converged solution. Also, the number of simulations that did not converge with the given CFL number are shown.

Additional to that, for back-step (1) and back-step (2) two extra networks will be considered. One of these back-steps is trained only on data points from back-step (1) in table 4.2, and the other is trained only on data points from back-step (2) in table 4.2. The results for all three networks will be compared to each other and the performance of the two CFL numbers in COMSOL. The network trained on back-step (1) will be called DS1 from “data set 1” and the network trained on back-step (2) will be called DS2. For simulations of back-step (3) and (4) only the network trained on the combination of data from all the back-steps will be considered.

The results on back-steps (1) and (2) for the network trained on the whole data set and the two networks that are trained on either data created from back-step (1) or back-step (2) are given table 5.5. In this table, we can see that the network trained on the combined data set performs in most cases better than the other two networks. It also converges for more simulations compared to the other two networks.

For back-step (1), data set 1 performs better than data set 2, which is expected. Still, the network trained on the combined data set outperforms these two significantly in terms of convergence and convergence acceleration.

For back-step (2), it was expected that the network trained on this simulation, DS2, would perform better than the network trained on back-step (1). However, both networks do not converge for 8 simulations, and DS1 is even the best option for more simulations than DS2. Still, the network on the combined data set is the best performing network for this simulation.

For the network trained on back-steps (1) to (4), all the results on back-steps (1) to (4) are given in table 5.6. For back-steps (1) and (3), the network performs really well, as it is the best option for most of the researched simulations. For back-step (2), the network performs equally well compared to one of the COMSOL CFL numbers in most cases, and in 3 of 24 cases it is the best option.

Unfortunately, the network is not able to predict CFL numbers for back-step (4) to obtain a converged solution. In 15 out of 16 simulations it is not able to converge. In the other simulation, it is the best option. So, although the simulation velocity and mesh size are within the range of the training data in table 4.2, the network is not able to predict good CFL numbers for this simulation. It is possible that this simulation is too difficult for the network, since the pseudo time-stepping that uses the COMSOL CFL numbers also barely obtains converged solutions. However, this is also the case for back-step (3). So, it is difficult to conclude why the network is struggling with back-step (4).

Back-step (1), rotated						
	CFL_{iter}	CFL_{e}	Network	Equal	None	Total
Best	1	2	14	4	3	**24**
No Convergence	9	6	6	-	-	-

Back-step (1), mirrored						
	CFL_{iter}	CFL_{e}	Network	Equal	None	Total
Best	1	2	14	4	3	**24**
No Convergence	9	6	6	-	-	-

Table 5.7: Summary of the results on back-step (1) with different geometry in table D.7. For each CFL number, the amount of simulations on which it performed best is given. That is, the CFL number that required the least amount of iterations to obtain a converged solution. Also, the number of simulations that did not converge with the given CFL number are shown.

Couette flow						
	CFL_{iter}	CFL_{e}	Network	Equal	None	Total
Best	0	0	45	0	0	**45**
No Convergence	1	1	0	-	-	-

Table 5.8: Summary of the results on the Couette simulation in table D.8. For each CFL number, the amount of simulations on which it performed best is given. That is, the CFL number that required the least amount of iterations to obtain a converged solution. Also, the number of simulations that did not converge with the given CFL number are shown.

Back-step rotated and mirrored

In order to see if the neural network is also able to generalise on different back-steps, back-step (1) of table 4.2 is rotated 90 degrees anti-clockwise and mirrored in the y-axis. The results are shown in table 5.7. In this table, we see that the network performs equally for both simulations. This is expected, since the two simulations are the same. However, if we compare these results with the results in table 5.6, we can see that the networks performs slightly better on the geometry on which the training data is obtained, as it is in one more case performing better than the COMSOL CFL numbers. So, apparently training on a back-step of a certain geometry does influence the networks predictions. Still, the network is able to predict CFL numbers for the mirrored and rotated back-steps such that convergence is accelerated in most cases shown in table 5.7. Only in three cases of these simulations, one of the COMSOL CFL numbers is the better option.

5.2.2. Results on other simulations

We already saw that the network performs well on different back-step simulations in COMSOL. However, since the network training data is created from different back-steps, we cannot conclude yet that the network is generalisable. In order to show that the network is able to predict CFL numbers for other laminar flow simulations, we will apply the network to different laminar flow simulations with velocities and mesh sizes within the range of the training data set.

First, we will discuss the outcomes of the networks applied to a Couette cylinder flow. Then, a Cavity flow is considered. Both of these simulations are described in Appendix A. Lastly, an obstacle will be positioned inside back-step (1) of table 4.2 and the Couette flow.

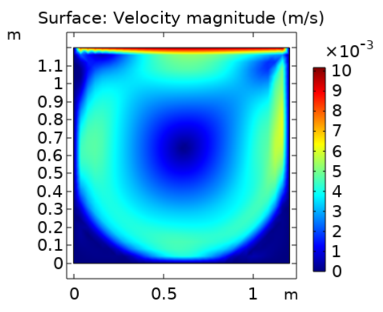
Couette flow

The network has been applied to different Couette cylinder flows. The first simulation has a inner cylinder with radius 0.2 m and outer cylinder with radius 0.4 m, and the second simulation is a scaled version, where the geometry and mesh sizes are divided by 5 and the flow is multiplied by 5. The geometry and simulation setup can be found in appendix A.

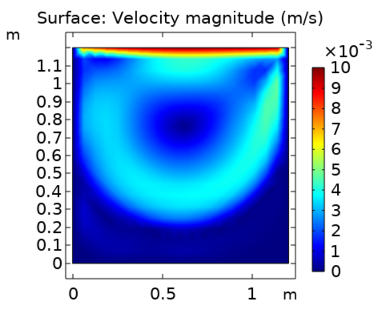
In table 5.8 the summary of the results of the network applied to the Couette flow are given. As is shown in this table, the network is able to accelerate convergence of the simulation for all the different Couette flows. Thus, the network generalises well for the Couette flow.

Cavity flow

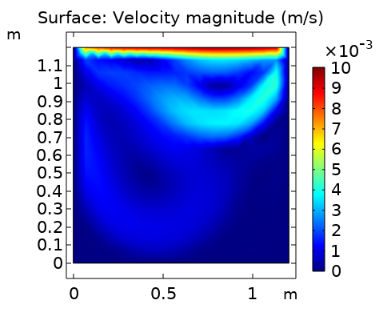
Another well known laminar flow model is the cavity flow, with geometry and physics shown in appendix A. When initially the network was applied to some cavity flow models, the network was not able to result



(a) Stable solution.



(b) Unstable solution.



(c) Unstable solution.

Figure 5.1: Different “converged” solutions of a cavity flow with wall velocity of 0.01 m/s, each with different mesh size.

in a converged solution. The largest network with 4 layers and 256 neurons was, however, able to result in a converged solution. Nevertheless, this solution was an unstable. Different unstable and a stable solution are shown in figure 5.1. In COMSOL, when the mesh size is too large, the solution tends to converge to an unstable solution.

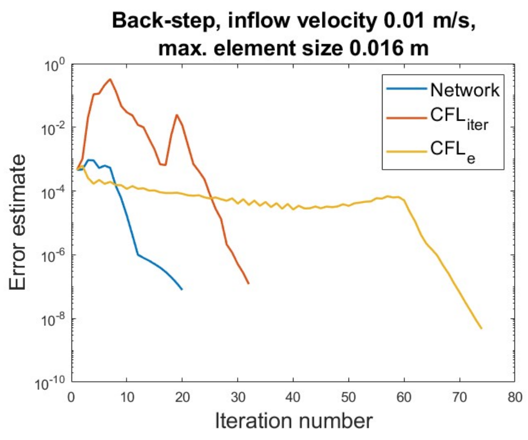
For different varieties of cavity flows with different mesh size and wall velocity, the network with 2 layers and 16 neurons was not able to lead to convergence and the largest network always showed an unstable solution if it led to convergence. Therefore, a different approach is used to generate results for this simulation.

In order to obtain the stable solution for the cavity flows, the initial solution that the network obtained was set to the solution after n iterations instead of the solution after 1 iteration. With this initial value, both networks were able to lead to the stable solution with less iterations than the default CFL numbers in COMSOL.

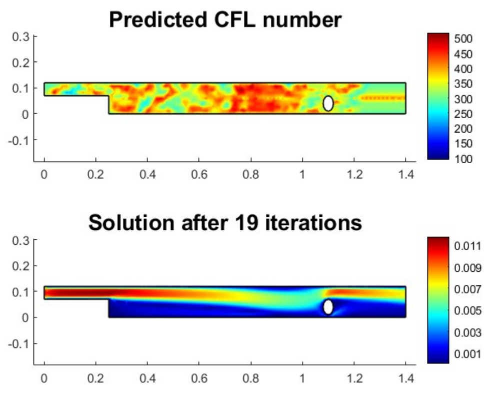
The number of iterations that are run with pseudo time-stepping with CFL$_e$ depends on the cavity simulation. For 10 different cavity flow simulations, the iteration before the network was applied were determined such that the network predictions led to convergence and the stable solution was obtained. The results are shown in table D.9 in appendix D, where we can see that the network with initial iterations by COMSOL still requires less iterations compared to the two CFL numbers in COMSOL.

Back-step and Couette flow with obstacle

For both back-step (1) and the Couette flow, the network was able to accelerate convergence. Even in some cases the solver converged where pseudo time-stepping with the CFL numbers in COMSOL did not converge. To challenge the network a bit more, different obstacles in different positions of the

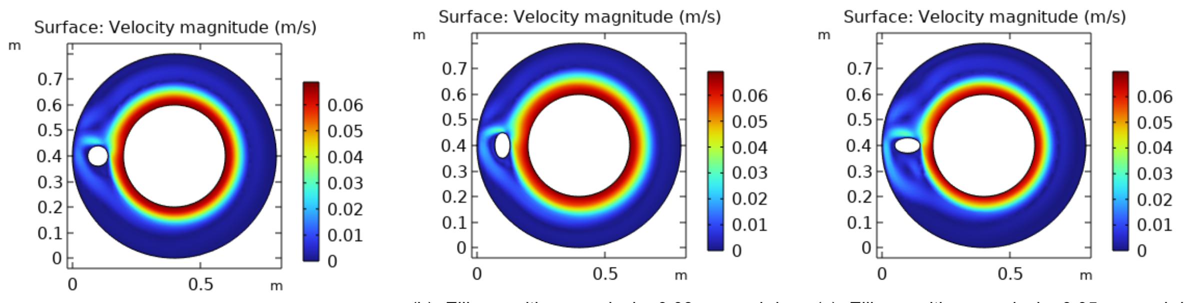


(a) Required number of iterations for the network prediction and the two CFL numbers in COMSOL.



(b) Last predicted CFL number with corresponding converged solution.

Figure 5.2: The required number of nonlinear iterations for each CFL number on the left and the CFL prediction with solution on the right. Back-step (1) has inflow velocity of 0.01 m/s and max. element size 0.016 m.



(a) Circle as an obstacle.

(b) Ellipse with a-semiaxis 0.03 m and b-semiaxis 0.05 m.

(c) Ellipse with a-semiaxis 0.05 m and b-semiaxis 0.03 m.

Figure 5.3: Couette flow with different kinds of obstacles. The wall rotation is 0.07 m/s and the maximum element size is 0.02 m. The center of the obstacle is x = 0.1 m and y = 0.4 m.

back-step and Couette flow COMSOL simulations are considered to check if the network predictions will also lead to convergence for these cases.

First, the back-steps with obstacle will be discussed. In the back-step three different obstacles will be considered: a circle with radius of 0.03 m, an ellipse with a-semiaxis 0.03 m and b-semiaxis 0.02 m, and an ellipse with a-semiaxis 0.02 m and b-semiaxis 0.03 m. For all the back-steps with obstacles, the inflows velocity is set on 0.008 m/s and the maximum mesh size is kept on 0.0126.

In table 5.9, the results for both simulations with obstacles are given. For the back-step, the network was the best option for 22 of 31 simulations, and was able to obtain a converged solutions for all the considered simulations. Only for one simulation CFL$_{iter}$ was the best option. With these results it can be concluded that the network is able to predict CFL numbers for back-steps with different obstacles, such that convergence is accelerated. Furthermore, in figure 5.2, the predicted CFL number and the required number of iterations is shown. In 5.2a, you can see that the network CFL performs much better than the two CFL numbers in COMSOL.

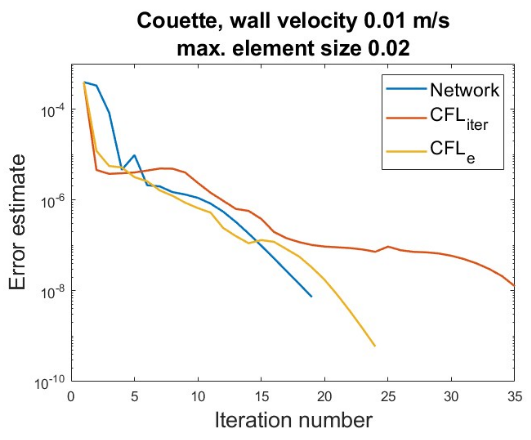
Secondly, the Couette flow with obstacle will be discussed. Again three different obstacles are considered, but this time the location of the obstacles are fixed. The obstacles are a circle with a radius of 0.04 m, an ellipse with a-semiaxis 0.03 m and b-semiaxis 0.05 m, and an ellipse with a-semiaxis 0.05 and b-semiaxis 0.03 m. The center of all these obstacles is located on x = 0.1 m and y = 0.4 m. The simulation is run with different maximum element sizes and wall velocities. The smaller, inner cylinder is rotating. The Couette flow with the obstacles are shown in figure 5.3.

In table 5.9, we can see that the network is the best option for 33 simulations out of the 45 simulations considered. For 10 simulations, it is not able to predict CFL numbers such that the solution converges. For all of the cases that the network does not lead to convergence, one of the two COMSOL CFL numbers also do not lead to a converged solution. In most cases that the network is the best option,

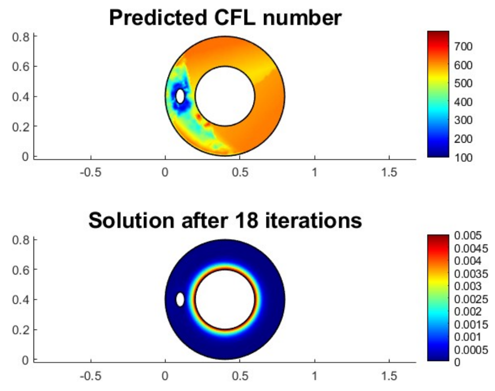
Back-step (1) with obstacles						
	CFL$_{iter}$	CFL$_e$	Network	Equal	None	Total
Best	1	0	22	8	0	**31**
No Convergence	10	0	0	-	-	-

Couette with obstacles						
	CFL$_{iter}$	CFL$_e$	Network	Equal	None	Total
Best	8	0	33	0	4	**45**
No Convergence	8	32	10	-	-	-

Table 5.9: Summary of the results on back-step (1) and Couette simulation with obstacles in tables D.10 and D.11. For each CFL number, the amount of simulations on which it performed best is given. That is, the CFL number that required the least amount of iterations to obtain a converged solution. Also, the number of simulations that did not converge with the given CFL number are shown.



(a) Required number of iterations for the network prediction and the two CFL numbers in COMSOL.



(b) Last predicted CFL number with corresponding converged solution.

Figure 5.4: The required number of nonlinear iterations for each CFL number on the left and the CFL prediction with solution on the right. The Couette flow has wall velocity of 0.01 m/s and max. element size 0.02 m.

the required nonlinear iteration count for convergence is (more than) halved compared to the two CFL numbers given in COMSOL. Thus, also for obstacles in the Couette simulation, the network is able to predict CFL numbers to accelerate convergence.

In figure 5.4 the predicted CFL number and the required number of iterations is shown for a Couette flow with obstacle. In 5.4a you can see that the network is able to converge in less iterations than the two CFL numbers in COMSOL.

6 Conclusion

This research aimed to improve the pseudo time-stepping algorithm in COMSOL by creating a generalisable neural network that can predict optimal CFL numbers. The chosen network has 2 hidden layers with each 16 neurons and is trained using previously computed optimal CFL numbers. In order to achieve generalisability, local data was used along with data points that contained information of a patch of elements. After applying the network to several different laminar flow simulations in COMSOL, it can be concluded that the network is in most cases able to accelerate convergence for a large range of laminar flow problems. Therefore, the network both improves the pseudo-time stepping algorithm and is generalisable.

In order to achieve this result, several approaches were considered. Firstly, a network was created that uses a loss computed directly from an error estimate of the COMSOL simulation. To the best of our knowledge, such a loss has not been used before, thus looking into this different approach could bring new insights to researching solver parameter prediction. However, the preliminary results disappointed and due to time limitation no further investigation was put into this network. Still, if it is possible to define a loss that is a direct representation of how well the COMSOL solver, or any other software, performs with the network prediction, it would be a new approach for predicting solver parameters with machine learning.

Secondly, a network that used an optimised CFL number as a target was considered. This network trained well with data from back-step simulations and was able to predict CFL numbers that accelerated convergence for other laminar flow simulations. However, the network is now only trained on data from several different back-step simulations. It would be interesting to see if data from other laminar flow simulations can be created to add to the training data set, and if training with this data set will improve generalisability or performance of the network. Nevertheless, this network that has only been trained on one kind of laminar flow model is generalising very well already.

Creating a neural network for finding optimal CFL numbers for pseudo time-stepping has not been studied before within the literature of predicting solver parameters with machine learning. Also, not yet a lot of research has been performed on the subject of predicting nonlinear solver parameters with machine learning until the present, especially not on generalisable machine learning methods. Being able to create a network, trained on optimised CFL targets, that is able to generalise and accelerate convergence, is a promising result for further research within this field.

6.1. Further research

Train network for turbulent flow and pseudo time-stepping

In this thesis only laminar flow problems have been considered. A natural extension for the research is to apply machine learning to predict CFL numbers for turbulent flow.

For example in [19], a residual smoothing technique is used to accelerate pseudo time-stepping, in this research called pseudo-transient continuation. It would be interesting to see how machine learning might help to improve pseudo time-stepping for these flow problems.

Recurrent neural networks
As mentioned in chapter 2, in the articles [17, 18] a recurrent neural network is used. These recurrent neural networks have a memory of the previous outputs and are more effective for long term dependencies. As the CFL number prediction might improve if the previous predictions are also considered, further investigating RNN's is an interesting approach for further research on this topic.

Furthermore, since CFL_{iter} in equation (3.7) depends on the iteration count and CFL_{e} in equation (3.8) depends on the previous CFL number, adding information about the previous CFL numbers combined by the amount of iterations that were already performed, an RNN can supply this extra information which cannot be used as an input to the network described in this research.

Create training data from different simulations
For this research, only data from different back-step simulations are used to create training data. Initially, also data from the cavity flow simulation would be used to train the network. However, a network trained on data points from the cavity flow only was not able to lead to convergence for cavity flow simulations on which it was trained.

There are two possible explanations for this. First, the cavity flow contains two singularities in the top two corners, making the residuals in those points very large compared to the residuals on the rest of the geometry. Taking the log of the residuals also did not help to improve the network training and predictions.

Second, unlike the back-step simulation, the cavity flow does not have a global optimum for the CFL. For the back-step, this global optimum was used as an initial guess for the CFL optimisation. With different initial guesses used to optimise local CFL numbers for the Cavity flow, each outcome would be completely different.

With these two reasons, COMSOL might not be able to optimise the CFL numbers for the cavity flow that result in improvement of the CFL number for pseudo time-stepping. Although in this research only data from back-step simulations is used, which showed promising results, adding information of other simulations could lead to even better results regarding generalisation and acceleration of the convergence.

Removing redundant input data
When using patches for the input data, the three elements around the central element have two vertices in common with the central element. As already explained in section 4.3, the redundant information of these double vertices is not removed. This double information makes the input data unnecessarily complex.

It is, however, difficult to determine how to order these vertices with the possibilities given with the COMSOL Matlab livelink. When obtaining mesh information from COMSOL in Matlab, all the vertices are labeled with a number and their coordinates are given. Yet, it is not clear how the vertices are labeled. So, in order to structure the vertices in a certain way, the coordinates of the elements should also be considered. Further research has to show how to remove the redundant information and how much simplifying the input data will improve the network training process and predictions.

Combining network and COMSOL CFL numbers
Shown at the results section, for the investigated COMSOL laminar flow simulations the network performed well compared to the two CFL numbers given in COMSOL. There were some cases, however, where the network works equally well or worse. Combining the network predictions and the COMSOL CFL numbers can maybe improve the overall effectiveness of the network.

For example, when the network prediction leads to a higher error estimate, one of the two CFL numbers defined in COMSOL can be used in order to accelerate the convergence even more. Or in the case of the cavity flow, first some pseudo time-step iterations are performed before the network was applied to the simulation. This in order to obtain the correct solution in less pseudo time-step iterations compared to the situation where only one of the CFL numbers in COMSOL was used.

Looking into a construction to combine the COMSOL CFL numbers with the neural network prediction can improve and accelerate convergence. Thus, it is an interesting topic to look into for further research.

References

[1] Sanjukta Bhowmick, Brice Toth, and Padma Raghavan. “Towards low-cost, high-accuracy classifiers for linear solver selection”. In: *International Conference on Computational Science*. Springer. 2009, pp. 463–472.

[2] Sanjukta Bhowmick et al. “Application of alternating decision trees in selecting sparse linear solvers”. In: *Software Automatic Tuning*. Springer, 2011, pp. 153–173.

[3] Sanjukta Bhowmick et al. “Application of machine learning to the selection of sparse linear solvers”. In: *Int. J. High Perf. Comput. Appl* (2006).

[4] E. Borgonovo. *Sensitivity Analysis: An Introduction for the Management Scientist*. Cham: Springer International Publishing, 2016. Chap. 5. DOI: 10.1007/978-3-319-52259-3.

[5] *COMSOL - Software for Multiphysics Simulation*. URL: https://www.comsol.com/ (visited on 09/18/2022).

[6] *Define Custom Regression Output Layer - MATLAB & Simulink - MathWorks Benelux*. URL: https://nl.mathworks.com/help/deeplearning/ug/define-custom-regression-output-layer.html (visited on 09/21/2022).

[7] Erika Fuentes. “Statistical and machine learning techniques applied to algorithm selection for solving sparse linear systems”. In: (2007).

[8] Peter Gainsford. *COMSOL Multiphysics Reference Manual*. v. 5.6. COMSOL. Stokholm, Sweden, 2020.

[9] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. 1st ed. Sebastopol, CA: O’Reilly Media, Mar. 2017. ISBN: 9781491962268.

[10] Philip E. Gill, Walter Murray, and Michael A. Saunders. “SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization”. In: *SIAM Journal on Optimization* 12.4 (2002), pp. 979–1006. DOI: 10.1137/S1052623499350013. eprint: https://doi.org/10.1137/S1052623499350013. URL: https://doi.org/10.1137/S1052623499350013.

[11] Alexander Grebhahn et al. “Performance-influence models of multigrid methods: A case study on triangular grids”. In: *Concurrency and Computation: Practice and Experience* 29.17 (2017). e4057 cpe.4057, e4057. DOI: https://doi.org/10.1002/cpe.4057. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4057. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4057.

[12] America Holloway and Tzu-Yi Chen. “Neural Networks for Predicting the Behavior of Preconditioned Iterative Solvers”. In: *Computational Science – ICCS 2007*. Ed. by Yong Shi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 302–309.

[13] Frank P. Incropera and David P. de Witt. *Fundamentals of heat and mass transfer*. 2nd ed. New York: Wiley, 1985.

[14] Carl Kelley and David Leyes. “Convergence Analysis of Pseudo-Transient Continuation”. In: *SIAM Journal on Numerical Analysis* 35 (Aug. 1996). DOI: 10.1137/S0036142996304796.

[15] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511791253.

[16] *LiveLink for MATLAB User’s Guide*. v. 6.0. COMSOL. Stokholm, Sweden, 2021.

[17] Nils Margenberg et al. “A neural network multigrid solver for the Navier-Stokes equations”. In: *Journal of Computational Physics* (Jan. 2021), p. 110983. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2022.110983. URL: http://dx.doi.org/10.1016/j.jcp.2022.110983.

[18] Nils Margenberg et al. *Deep neural networks for geometric multigrid methods*. 2021. DOI: 10.48550/ARXIV.2106.07687. URL: https://arxiv.org/abs/2106.07687.

[19] Dimitri J. Mavriplis, Behzad Ahrabi, and Michael Brazell. “Strategies for Accelerating Newon Method Continuation in CFD Problems”. In: ().

[20] Pate Motter et al. “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.

[21] *Normalize data - MATLAB normalize - MathWorks Benelux*. URL: `https://nl.mathworks.com/help/matlab/ref/double.normalize.html` (visited on 09/18/2022).

[22] Deep Ray and Jan S. Hesthaven. “Detecting troubled-cells on two-dimensional unstructured grids using a neural network”. In: *Journal of Computational Physics* 397 (2019), p. 108845. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/j.jcp.2019.07.043`. URL: `https://www.sciencedirect.com/science/article/pii/S0021999119305297`.

[23] Vinícius Luiz Silva et al. “Machine learning acceleration for nonlinear solvers applied to multiphase porous media flow”. In: *Computer Methods in Applied Mechanics and Engineering* 384 (Oct. 2021), p. 113989. DOI: `10.1016/j.cma.2021.113989`.

[24] *Turbulent flow over a backward-facing step*. URL: `https://www.comsol.com/model/turbulent-flow-over-a-backward-facing-step-228` (visited on 03/17/2022).

[25] J. Watt, R. Borhani, and A. Katsaggelos. *Machine Learning Refined: Foundations, Algorithms and Applications*. Cambridge: Cambridge University Press, 2016.

[26] P. Wesseling. *Elements of Computational Fluid Dynamics*. Delft: TU Delft, 2014.

A
COMSOL simulation setup

In this section the simulation setup of the back-step and the cavity flow in COMSOL will be discussed, such that the model can be reproduced in COMSOL by the reader. The geometry, mesh and elements and the study steps will be explained.

Back-step

The geometry of this model is shown in figure A.1. The inflow boundary is located at the top left edge coloured in red, and the flow moves towards the right outflow boundary coloured in blue. Due to the increase of cross-sectional area, the laminar flow, the Reynolds number increases and flow separation can occur [24]. Laminar flow is more likely to occur with higher flow velocities and a larger difference in cross-sectional area.

Geometry and initial conditions

There are several possibilities for the geometry and the initial conditions when creating a backward-facing step simulation. The chosen geometry does not converge with the default solver settings, but it does when pseudo time-stepping is used.

The dimensions of the backwards-facing step flow simulation that are used for creating the data set in section 4.3, are given in table A.1. In this table, the first back-step mentioned is the back-step with its converged solution in figure A.2. The second back-step in the table is a scaled version of the first back-step. With higher inflow velocities, the (cell) Reynolds numbers are the same for both simulations. So, by scaling, a larger range of velocities and mesh sizes are created for the data set. To also create a wider range of Reynolds numbers, the third and the fourth back-steps are created. The cross-sectional area is twice as large compared to the first two back-steps. Again, the fourth back-step is a scaled version of the third back-step.

Besides the Reynolds number, other factors that influence the simulation solution are the inflow velocity, the initial velocity and the mesh size. As mentioned earlier, by scaling two of the back-steps, the Reynolds number stays the same, but the inflow velocities can be higher with a smaller geometry and mesh. In this research, it was chosen to set the initial velocity on zero.

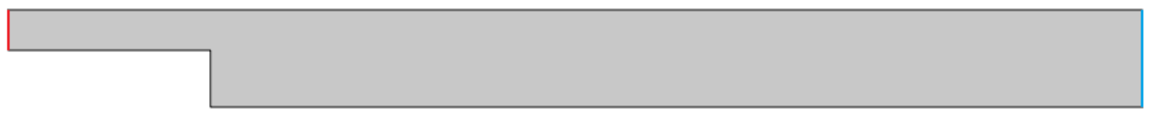


Figure A.1: The geometry back-step laminar flow simulation in COMSOL, with inflow boundary in red and outflow boundary in blue. The dimensions are given in Table A.1.

	Inflow tunnel Height&Width (m)	Tunnel after geometry enlargement Height&Width (m)
(1) Back-step	0.05×0.25	0.12×1.15
(2) Back-step scaled	0.005×0.025	0.012×0.115
(3) Back-step higher Re	0.08×0.25	0.22×1.15
(4) Back-step higher Re scaled	0.008×0.025	0.022×0.115

Table A.1: Dimensions of the back-step simulations in COMSOL

Mesh and elements

For this model we use triangular elements. In section 4.3, we saw that the information on the element vertices, edges and centroid are used as an input for the neural network. Only working with triangular elements makes it easier to have inputs for the neural network with the same size.

To create data, different meshes are created and used. All the created meshes are calibrated for fluid dynamics. For the back-step, the “extra coarse” mesh size is mostly chosen. To create a variety in element sizes, the maximum element size takes different values for different simulations. The default discretisation for laminar flow is used, that is P1+P1 (triangular) elements. The P1+P1 elements are piecewise linear elements for both the velocity and pressure [8].

Cavity flow

In the cavity flow COMSOL simulation, the geometry is a square where the top wall is moving in the x-direction. The geometry, initial conditions and the mesh of the cavity flow simulation will be discussed. The geometry of the cavity flow simulation is a square, where the top boundary is a moving wall in the $+x$-direction. A pressure point constraint is added to the top left vertex of the square, where the pressure p_0 is set to 0 Pa. This is to ensure the solution is unique.

For a wall movement of 0.09 m/s, the converged solution is given in figure A.3. Simulation with different wall movement velocities were run to create a data set with a larger range of velocities.

The mesh of the cavity flow is very important, because with a poorly chosen mesh the solver converges to an unstable solution. To prevent this, a mesh small enough is required to obtain the stable solution. Furthermore, the mesh can be chosen smaller at the moving wall boundary, to obtain a more precise solution at this boundary.

Again, triangular elements are used for the mesh since the network can only process triangular elements. Different meshes are created to create data sets with a larger range of element edge lengths. Again the mesh is calibrated with the default discretisation for laminar flow.

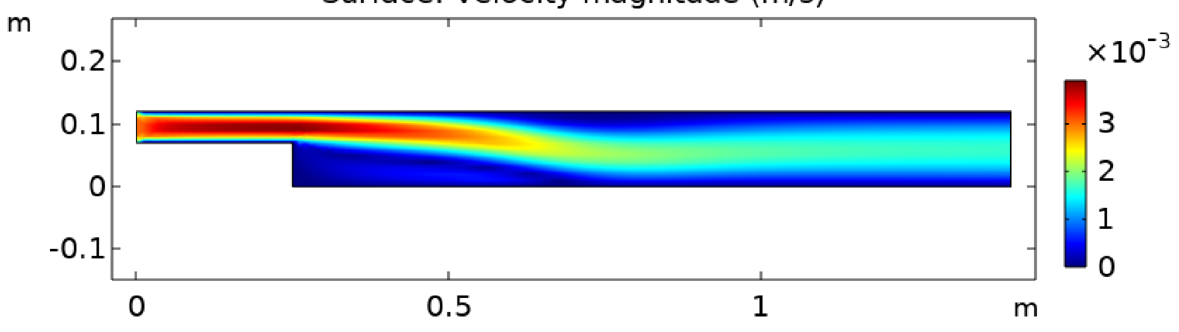


Figure A.2: Converged solution of a laminar flow simulation of the back-step model in COMSOL, with inflow velocity 0.003 m/s.

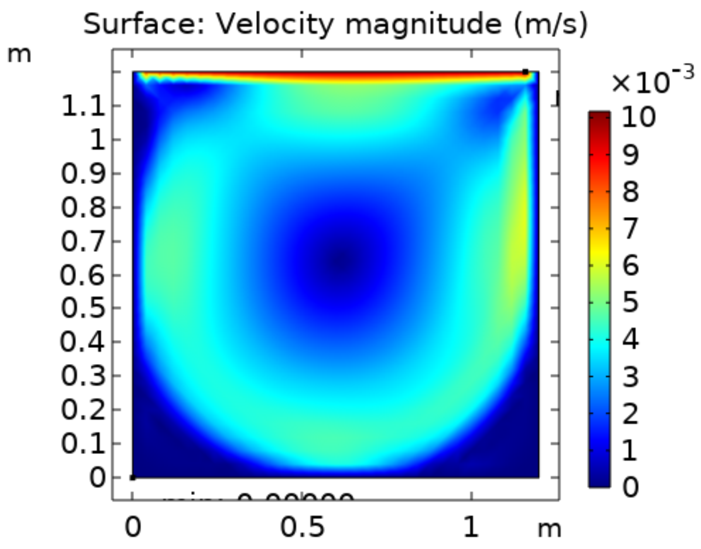


Figure A.3: Converged solution of a laminar flow simulation of a cavity flow in COMSOL with wall velocity 0.09 m/s.

Couette cylinder flow

A 2D Couette cylinder flow is also considered to investigate if the created neural network is generalizable. For this Couette flow, there are two cylinders, an outer, larger cylinder and an inner,smaller cylinder. In between the cylinders is water. The inner cylinder is rotating, enabling a sliding or tangentially moving wall with velocity `u_in`, a global variable speed of choice. Again a pressure point constraint is added to the inner cylinder, with p_0 = 0 Pa, to ensure that the solution is unique.

For a rotating wall with velocity of 0.1 m/s, the converged solution is given in figure A.4. To test generalizability of the network different wall velocities are chosen within the velocity learning range of the data set on which the network is trained.

The mesh of the Couette flow again consists of only triangular P1+P1 elements. The mesh generated by the predefined element size is sufficient to obtain a good, converged solution. To test generalizability of the network, different maximum element sizes were used.

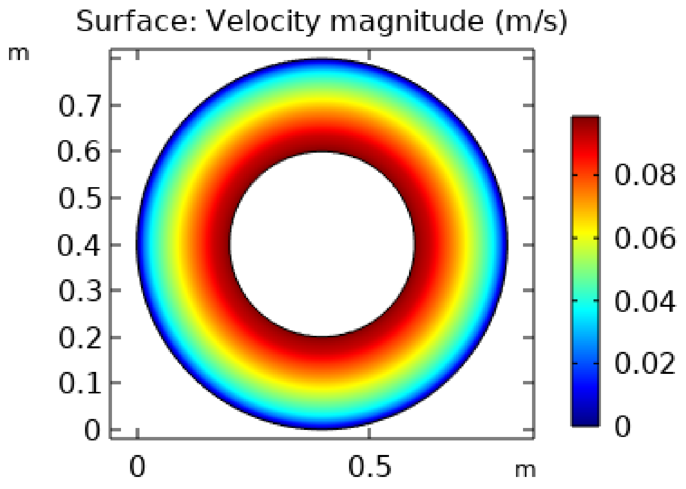


Figure A.4: Converged solution of a laminar flow simulation of a Couette cylinder flow in COMSOL with rotating wall velocity 0.1 m/s.

Study steps

An organized plan for the study step is a necessity, since there is a step required to compute the solution with the default pseudo time-step settings, a step that computes the solution with the predicted CFL number, a sensitivity analysis to compute necessary derivatives and an optimisation step to compute the optimal value for the CFL number.

First we will discuss the study step with the default settings, then the study step that uses the predicted CFL and how this step is connected to the first study step with default settings, and lastly we discuss the sensitivity analysis and optimisation step.

Study step with default settings

First, a study step is created to compute the solution for the first steps using the default settings for pseudo time-stepping. The information of the last step of this study will be used as an input for the neural network, or to create a data set to train the network with. The data set and network inputs will be discussed more thoroughly in section 4.3.

In the laminar flow menu, in advanced settings, the pseudo time-stepping for the stationary equation form is enabled with an automatic CFL number expression. This expression is shown in equation (3.7). The used solver is Newton constant with a damping factor of 1 and is terminated when a given number of iterations is reached.

The pseudo time-stepping is also used as a stabilisation and acceleration technique within the fully coupled stationary solver. The pseudo time-step algorithm is explained in section 3.3 Again, the default settings for this method are used.

Study step with CFL prediction

After the study step that performs an iteration with the default settings, the next study step will be performed with the predicted CFL number by the neural network. The initial solution for this step is obtained by the last solution of the study step with default settings for the pseudo time-stepping after one iteration. At the next iterations, the initial solution will be set to the previously obtained solution by its own study step.

Pseudo time-stepping for laminar flow models in COMSOL [8] is given by

$$\rho \frac{\mathbf{u} - \text{nojac}(\mathbf{u})}{\Delta \tilde{t}} + \rho(\mathbf{u} \cdot \nabla)\mathbf{u} = \nabla \cdot [-p\mathbf{I} + \mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)] + \mathbf{F}. \tag{A.1}$$

Again, $\Delta \tilde{t}$ is the pseudo time-step. The term $\mathbf{u} - \text{nojac}(\mathbf{u})$ is a special term defined in COMSOL that is zero.

In order to include the CFL number obtained from the neural network in Matlab, an extra laminar flow interface is created to include this CFL number in COMSOL. First, an interpolation function is defined. This interpolation function `int1(x,y)` reads the output of the neural network combined with its coordinates from a text file and interpolates these values over the domain. Then, a weak contribution is added to the new laminar flow interface, to function as the additional term in pseudo time-stepping. The weak expression is given as

```
spf.rho*(-(u-nojac(u2))*test(u)/cfl2-(v-nojac(v2))*test(v)/int1(x,y)).
```

Note again that `int(x,y)` is the network prediction. The fully coupled solver computes one Newton constant step, with damping factor 1, with the predicted CFL number for pseudo time-stepping.

Optimisation

The neural network has to be trained so that it can predict the CFL number that leads to the smallest loss. This optimised CFL number will be used as a target to train the network in section 4.4.2, and the background information of the optimisation is explained in section B.

The general optimisation study step minimises the value of a given global objective for a given control variable field. The global objective is the loss function given in equation (4.2) and the control variable is the CFL number. The initial value for the CFL number is set to 300, the optimisation method used is SNOPT with a tolerance of 1E-13 and maximum number of evaluations set to 100,000.

Sensitivity analysis

In order to compute the derivatives necessary for the training of the network in section 4.4.1, a sensitivity analysis is performed. In appendix B the mathematical background of the sensitivity analysis is explained within the frame of the COMSOL computation step.

The derivative that has to be computed in section 4.4.1, is the derivative of the loss function in equation (4.2). That means that the global objective function will equal this loss function. The exact solution is computed in a separate study with its own defined laminar flow physics in the COMSOL simulation. So before the sensitivity can be computed, this study step has to be computed in order to obtain the “exact” or converged solution.

The control variable name will be `CFL`, with initial value `int1(x,y)`, which is the interpolated predicted CFL number from the neural network.

B Sensitivity analysis and optimisation in COMSOL

Sensitivity analysis in COMSOL

In order to obtain the gradients for error estimate in (4.2), a sensitivity analysis in COMSOL has to be performed. The sensitivity interface in COMSOL does not contain any physics, but it evaluates the sensitivity of the model with respect to a given variable [8].

Evaluating the sensitivity of a function $F(x)$ with respect to the variable x at a given point x_0 means computing the derivative $\partial F/\partial x$ at the point x_0 [8]. We want to obtain the derivative of the error estimate with respect to the predicted local CFL number:

$$\frac{\partial}{\partial \mathrm{CFL}_{\mathrm{pred}}}\left(\int_{\Omega}(u-u_{ex})^2+(v-v_{ex})^2\,\mathrm{d}\Omega\right)^{\frac{1}{2}}.$$

The solutions u and v are in this case dependent of the CFL prediction. The global objective of the sensitivity is the error estimate and the control variable field is the CFL prediction.

When a sensitivity analysis is performed for a stationary problem, the sensitivity of a functional

$$Q = Q(u_p, p),$$

with respect to the sensitivity variables p are computed. Here, u_p is the so-called forward solution to the parameterised discrete forward problem given by

$$L(u_p, p) = N_F\Lambda_p \qquad\qquad M(u_p, p) = 0,$$

where Λ_p are the constraint Lagrange multipliers corresponding to the constraints given by M, and N_F is the constraint force Jacobian. Here it is assumed that Q does not explicitly depend on Λ_p [4].

To compute the sensitivity of Q with respect to p, or compute the derivative with respect to p, the chain rule is applied:

$$\frac{d}{dp}Q(u(p),p) = \frac{\partial Q}{\partial p}+\frac{\partial Q}{\partial u}\frac{\partial u}{\partial p}. \tag{B.1}$$

In this expression the term $\partial u/\partial p$ is not known. This is where the forward problem will be used. The forward problem is differentiated with respect to p:

$$-\frac{\partial L}{\partial u}\frac{\partial u_p}{\partial p}-\frac{\partial M}{\partial u} = \frac{\partial L}{\partial P}-\left(\frac{\partial N_F}{\partial u_p}\frac{\partial u_p}{\partial p}+\frac{\partial N_F}{\partial p}\right) \qquad\qquad N_F\frac{\partial u_p}{\partial p} = \frac{\partial M}{\partial p}.$$

Because of linear constraints, or $\partial N_F/\partial u = 0$, the first term inside the parentheses above is assumed to be zero. When furthermore is assumed that N_F is also independent of p, the problem above can be rewritten in matrix form:

$$J\begin{pmatrix} \frac{\partial u_p}{\partial p} \\ \frac{\partial \Lambda_p}{\partial p} \end{pmatrix} = \begin{pmatrix} \frac{\partial L}{\partial p} \\ \frac{\partial M}{\partial p} \end{pmatrix} \qquad J = \begin{bmatrix} -\partial L/\partial u & N_F \\ -\partial M/\partial u & 0 \end{bmatrix}$$

This equation is then solved for $\partial u_p/\partial p$ and $\partial \Lambda_p/\partial p$ and the solutions are plugged in into equation (B.1), which can then be solved:

$$\frac{dQ}{dp} = \frac{\partial Q}{\partial p} + \begin{pmatrix} \partial Q/\partial u \\ 0 \end{pmatrix}^{\mathrm{T}} J^{-1} \begin{pmatrix} \partial L/\partial p \\ \partial M/\partial p \end{pmatrix}.$$

By solving this equation, the sensitivity dQ/dp can be computed [4].

optimisation in COMSOL

The neural network has to be trained so that it can predict the CFL number that leads to the smallest loss. In order to understand which local CFL number leads to the smallest loss in COMSOL, an optimisation study is performed within COMSOL.

The general optimisation study step minimises the value of a given global objective for a given control variable field. The global objective is the error estimate loss in 4.2 and the control variable is the CFL number, the optimal global CFL number computed in section 4.2.1. The initial value for the CFL number is set to 300, the optimisation method used is SNOPT (sparse nonlinear optimiser) with a tolerance of 10^{-13} and maximum number of evaluations set to 100,000. The target is to optimise the CFL number such that the solution after one step with this CFL number is closest to the converged solution.

SNOPT is a sequential quadratic programming (SQP) method, which has been proved as highly effective as an optimiser for problems with smooth, nonlinear and convex functions in both the objective and the constraints [10]. Consider the optimisation problem of the form

$$\begin{aligned} &\underset{x\in\mathbb{R}^n}{\text{minimise}} && f(x) \\ &\text{subject to} && l \leq \begin{pmatrix} x \\ c(x) \\ Ax \end{pmatrix} \leq u. \end{aligned} \tag{B.2}$$

Here, $f(x)$ is the linear or nonlinear objective function we want to minimise, $c(x)$ is a vector of nonlinear constraints with sparse derivatives, A is a sparse matrix with linear constraints and l and u are respectively the lower and upper bounds [10]. SNOPT is an iterative optimisation method that uses major and minor iterations.

First infeasible linear constraints are detected by solving a problem of the form

$$\begin{aligned} &\underset{x,v,w}{\text{minimise}} && \|v+w\|_1 \\ &\text{subject to} && l \leq \begin{pmatrix} x \\ Ax - v + w \end{pmatrix} \leq u \quad v \geq 0, w \geq 0. \end{aligned}$$

Where v and w are linear constraints, which are handled implicitly. If these linear constraints are infeasible, i.e. $v = 0$ and $w = 0$, SNOPT will terminate the computation [10].

If it is feasible, it will then move on to solve problem (B.2) using quadratic-programming (QP) subproblems. For each major iteration, a QP subproblem is solved by an iterative process. The iterations of the subproblem are the minor iterations. If a QP subproblem is not feasible or not bounded, SNOPT will solve a so-called “nonlinear elastic” mode problem:

$$\begin{aligned} &\underset{x,v,w}{\text{minimise}} && f(x) + \gamma\|v + w\|_1 \\ &\text{subject to} && l \leq \begin{pmatrix} x \\ Ax - v + w \end{pmatrix} \leq u \quad v \geq 0, w \geq 0. \end{aligned}$$

With $\gamma \geq 0$ a penalty parameter. If problem (B.2) has a feasible solution, and γ is sufficiently large, the solution of the nonlinear elastic mode problem will be similar to (B.2). However, if (B.2) does not have a feasible solution, but γ is again sufficiently large, the nonlinear elastic mode problem gives a good estimation of the solution of (B.2) [10].

C
Data points from patch or element

In Section 4.3 both data from a single element and data from a patch of elements is discussed. The data from one single element is located on each of the element vertices, and from a patch on all the vertices of the four elements. In [17, 18], the input data is created from patches. In this study, also patches of elements are used for the input data for the neural network.

In order to investigate if patches of four elements will perform better as input data compared to data obtained from only one element, two networks are trained and their performance will be compared. Back-step (1) from A.1 and the Couette flow with outer cylinder with radius 0.4 m and inner cylinder with radius 0.2 m will be considered.

The results for the performance on back-step (1) are given in table C.1. As we see the patch is the best option in 6 cases, the element data in 5 cases and in 12 cases the two networks perform the same. Only for 1 simulation, neither the networks nor the CFL numbers in COMSOL were able to obtain a converged solution.

In two cases where the the patch data was better than the element data, the network trained on the element data did not converge. The other way around was not the case, so the network trained on the patch data always converged when the network trained on the element data was the best option.

The results on the Couette flow are given in table C.1. For the 30 simulations, in 14 cases the results of both networks are equal, in 9 cases the network trained on element patches performs better and in 6 cases the network trained on data from a single element was the best option. Only in one case both networks did not meet the convergence criterion.

In three cases that the patch network outperformed the element network, the element network would not converge at all. The other way around was only one case. This given the fact that the patch also is the better option in more cases compared to data from a single element, it can be concluded that the patch network generalizes better.

For both cases, the networks trained on a patch of elements or on single element data performed equally well in most cases. However, the network trained on the patch data did converge in more cases, compared to the network trained on element data. That is why a network trained on data generated from a patch of elements is used for this research.

Back-step (1)							
	CFL$_{iter}$	CFL$_e$	Patch	Element	Equal	None	Total
Best	0	0	6	5	12	1	**24**
No Convergence	6	3	1	4	-	-	-
Couette flow							
	CFL$_{iter}$	CFL$_e$	Patch	Element	Equal	None	Total
Best	0	1	9	6	14	0	**30**
No Convergence	1	1	2	4	-	-	-

Table C.1: Summary of the results on back-step (1) and Couette simulations for patch and element data in tables C.2 and C.3. For each CFL number, the amount of simulations on which it performed best is given. That is, the CFL number that required the least amount of iterations to obtain a converged solution. Also, the number of simulations that did not converge with the given CFL number are shown.

Back-step (1)						
Max. element size	Inflow velocity	Iterations CFL$_{iter}$	Iterations CFL$_e$	Iterations Patch	Iterations Element	Best
0.0106	0.001	6	15	6	6	Equal
0.0156	0.001	6	13	6	6	Equal
0.0206	0.001	6	11	6	6	Equal
0.0256	0.001	6	10	6	6	Equal
0.0106	0.004	150	65	7	10	Patch
0.0156	0.004	24	50	11	11	Equal
0.0206	0.004	17	36	12	13	Equal
0.0256	0.004	19	33	14	13	Equal
0.0106	0.007	150	105	10	-	Patch
0.0156	0.007	31	55	11	11	Equal
0.0206	0.007	26	60	16	16	Equal
0.0256	0.007	33	54	22	19	Element
0.0106	0.01	-	150	14	-	Patch
0.0156	0.01	64	78	19	21	Equal
0.0206	0.01	34	87	23	21	Equal
0.0256	0.01	48	75	24	22	Equal
0.0106	0.012	-	150	-	-	None
0.0156	0.012	150	96	22	76	Patch
0.0206	0.012	40	106	28	24	Element
0.0256	0.012	61	94	26	21	Element
0.0106	0.015	150	150	16	-	Patch
0.0156	0.015	115	129	23	31	Patch
0.0206	0.015	37	113	39	32	Element
0.0256	0.015	82	118	27	22	Element

Table C.2: Back-step (1) results with CFL numbers from COMSOL, for a network trained on element patches and for a network trained on data from a single element.

Couette flow, Outer cylinder radius 0.4 m, inner cylinder radius 0.2 m						
Max. element size	**Wall velocity**	**Iterations CFL$_{iter}$**	**Iterations CFL$_e$**	**Iterations Patch**	**Iterations Element**	**Best**
0.014	0.01	55	29	11	11	Equal
0.016	0.01	54	29	23	22	Equal
0.018	0.01	51	28	9	17	Patch
0.02	0.01	50	29	10	10	Equal
0.022	0.01	47	28	10	9	Equal
0.014	0.03	150	150	10	11	Equal
0.016	0.03	54	31	14	17	Patch
0.018	0.03	53	30	-	16	Element
0.02	0.03	49	28	14	17	Patch
0.022	0.03	48	25	16	18	Patch
0.014	0.04	61	37	13	-	Patch
0.016	0.04	55	33	21	16	Element
0.018	0.04	52	29	15	18	Patch
0.02	0.04	51	28	14	16	Equal
0.022	0.04	48	25	24	18	Element
0.014	0.05	57	35	22	22	Equal
0.016	0.05	62	38	14	14	Equal
0.018	0.05	53	30	16	16	Equal
0.02	0.05	51	28	16	16	Equal
0.022	0.05	47	25	27	-	Patch
0.014	0.07	57	35	17	17	Equal
0.016	0.07	55	33	17	18	Equal
0.018	0.07	53	30	21	16	Element
0.02	0.07	51	28	23	20	Element
0.022	0.07	47	24	23	61	Patch
0.014	0.1	57	36	17	19	Equal
0.016	0.1	55	33	25	21	Element
0.018	0.1	52	29	25	-	Patch
0.02	0.1	50	27	24	23	Equal
0.022	0.1	47	24	100	100	None

Table C.3: Couette flow simulation results with CFL numbers from COMSOL, for a network trained on element patches and for a network trained on data from a single element.

D
Result tables

Grid search, back-step (1)							
Max. element size	Inflow velocity	Iterations CFL$_{iter}$	Iterations CFL$_e$	Iterations Network 1	Iterations Network 2	Iterations Network 3	Best
0.0106	0.001	6	15	6	6	6	Equal
0.0156	0.001	6	13	6	6	6	Equal
0.0206	0.001	6	11	6	6	6	Equal
0.0256	0.001	6	10	6	6	6	Equal
0.0106	0.004	150	65	-	23	100	Net 2
0.0156	0.004	24	50	13	12	11	Equal
0.0206	0.004	17	36	15	12	12	Equal
0.0256	0.004	19	33	14	14	14	Equal
0.0106	0.007	150	105	-	-	100	None
0.0156	0.007	31	55	100	100	11	Net 3
0.0206	0.007	26	60	18	17	16	Equal
0.0256	0.007	33	54	22	22	22	Equal
0.0106	0.01	-	150	-	-	100	None
0.0156	0.01	64	78	100	100	19	Net 3
0.0206	0.01	34	87	27	27	23	Net 3
0.0256	0.01	48	75	26	25	24	Equal
0.0106	0.012	-	150	-	-	-	None
0.0156	0.012	150	96	15	100	22	Net 1
0.0206	0.012	40	106	33	32	28	Net 3
0.0256	0.012	61	94	27	27	26	Equal
0.0106	0.015	150	150	-	-	16	Net 3
0.0156	0.015	115	129	100	100	23	Net 3
0.0206	0.015	37	113	46	42	39	Net 3
0.0256	0.015	82	118	29	28	27	Equal

Table D.1: Back-step (1) with results from network 1, network 2 and network 3. In the last column the best option of the networks is shown and we see that in most cases network 3 is the best option. Network 1 has 4 hidden layers with each 256 neurons, network 2 has 3 hidden layers with each 64 neurons and network 3 has 2 hidden layers with each 16 neurons. The “-” means the solver diverged.

Grid search, Couette flow							
Max. element size	**Wall velocity**	**Iterations CFL$_{iter}$**	**Iterations CFL$_e$**	**Iterations Network 1**	**Iterations Network 2**	**Iterations Network 3**	**Best**
0.014	0.01	55	29	11	-	11	Equal
0.016	0.01	54	29	12	-	23	Net 1
0.018	0.01	51	28	25	-	9	Net 3
0.02	0.01	50	29	10	-	10	Equal
0.022	0.01	47	28	9	-	10	Equal
0.014	0.03	150	150	12	-	10	Equal
0.016	0.03	54	31	18	-	14	Net 3
0.018	0.03	53	30	17	-	-	Net 1
0.02	0.03	49	28	20	-	14	Net 3
0.022	0.03	48	25	15	-	16	Equal
0.014	0.04	61	37	24	-	13	Net 3
0.016	0.04	55	33	16	-	21	Net 1
0.018	0.04	52	29	21	-	15	Net 3
0.02	0.04	51	28	13	-	14	Equal
0.022	0.04	48	25	14	29	24	Net 1
0.014	0.05	57	35	15	-	22	Net 1
0.016	0.05	62	38	14	-	14	Equal
0.018	0.05	53	30	14	-	16	Equal
0.02	0.05	51	28	15	-	16	Equal
0.022	0.05	47	25	15	-	27	Net 1
0.014	0.07	57	35	17	-	17	Equal
0.016	0.07	55	33	13	-	17	Net 1
0.018	0.07	53	30	17	-	21	Net 1
0.02	0.07	51	28	19	-	23	Net 1
0.022	0.07	47	24	19	-	23	Net 1
0.014	0.1	57	36	19	-	17	Equal
0.016	0.1	55	33	20	40	25	Net 1
0.018	0.1	52	29	17	-	25	Net 1
0.02	0.1	50	27	21	16	24	Net 2
0.022	0.1	47	24	22	-	100	Net 1

Table D.2: Couette flow with results from network 1, network 2 and network 3. In the last column the best option of the networks is shown and we see that in most cases network 3 is the best option. The outer cylinder has radius 0.4 m and the inner, moving cylinder has radius of 0.2 m. Network 1 has 4 hidden layers with each 256 neurons, network 2 has 3 hidden layers with each 64 neurons and network 3 has 2 hidden layers with each 16 neurons. The “-” means the solver diverged.

Back-step (1)							
Max. element size	Inflow velocity	Iterations CFL$_{iter}$	Iterations CFL$_e$	Iterations Combined	Iterations DS 1	Iterations DS 2	Best Network
0.0106	0.001	6	15	6	6	6	Equal
0.0156	0.001	6	13	6	6	6	Equal
0.0206	0.001	6	11	6	7	6	Equal
0.0256	0.001	6	10	6	6	6	Equal
0.0106	0.004	150	65	100	22	-	DS 1
0.0156	0.004	24	50	11	14	12	Combined
0.0206	0.004	17	36	12	17	15	Combined
0.0256	0.004	19	33	14	19	18	Combined
0.0106	0.007	150	105	100	-	-	Combined
0.0156	0.007	31	55	11	-	31	Combined
0.0206	0.007	26	60	16	26	22	Combined
0.0256	0.007	33	54	22	32	29	Combined
0.0106	0.01	-	150	100	-	-	Combined
0.0156	0.01	64	78	19	41	60	Combined
0.0206	0.01	34	87	23	35	36	Combined
0.0256	0.01	48	75	24	49	43	Combined
0.0106	0.012	-	150	-	-	-	None
0.0156	0.012	150	96	22	53	100	Combined
0.0206	0.012	40	106	28	45	45	Combined
0.0256	0.012	61	94	26	60	53	Combined
0.0106	0.015	150	150	16	-	-	Combined
0.0156	0.015	115	129	23	100	100	Combined
0.0206	0.015	37	113	39	41	43	Combined
0.0256	0.015	82	118	27	74	63	Combined

Table D.3: Back-step (1) from table A.1, where a network trained on data from all the back-steps in table A.1 called “Combined”, trained on data from back-step (1) called “DS1” and trained on data from back-step (2) called “DS2”. The “-” means the solver diverged.

Back-step (2)							
Max. element size	Inflow velocity	Iterations CFL$_{iter}$	Iterations CFL$_e$	Iterations Combined	Iterations DS1	Iterations DS 2	Best Network
0.00106	0.01	6	15	6	6	6	Equal
0.00156	0.01	6	13	6	6	6	Equal
0.00206	0.01	6	11	7	7	7	Equal
0.00256	0.01	6	10	6	6	6	Equal
0.00106	0.04	-	128	24	-	23	DS2 & Combined
0.00156	0.04	24	85	62	23	28	DS1
0.00206	0.04	17	50	17	18	17	Equal
0.00256	0.04	19	38	19	20	19	Equal
0.00106	0.07	-	150	-	-	-	None
0.00156	0.07	45	82	-	45	-	DS1
0.00206	0.07	26	70	26	26	26	Equal
0.00256	0.07	33	61	32	34	32	Equal
0.00106	0.1	-	150	-	-	-	None
0.00156	0.1	34	77	35	100	100	Combined
0.00206	0.1	34	73	35	36	35	Equal
0.00256	0.1	48	66	49	48	49	Equal
0.00106	0.12	-	150	-	-	-	None
0.00156	0.12	150	96	89	100	100	Combined
0.00206	0.12	41	106	46	47	46	Equal
0.00256	0.12	61	94	59	61	60	Equal
0.00106	0.15	-	150	-	-	-	None
0.00156	0.15	77	129	100	100	100	None
0.00206	0.15	36	113	41	41	41	Equal
0.00256	0.15	82	118	71	82	75	Combined

Table D.4: Back-step (2) from table A.1, where a network trained on data from all the back-steps in tableA.1 called “Combined”, trained on data from back-step (1) called “DS1” and trained on data from back-step (2) called “DS2”. The “-” means the solver diverged.

Back-step (3)					
Max. element size	Inflow velocity	Iterations CFL$_{iter}$	Iterations CFL$_e$	Iterations network	Better?
0.0126	0.001	-	30	10	Yes
0.0156	0.001	-	26	12	Yes
0.0186	0.001	150	24	11	Yes
0.0206	0.001	150	22	10	Yes
0.0126	0.004	-	149	21	Yes
0.0156	0.004	-	124	43	Yes
0.0186	0.004	-	114	12	Yes
0.0206	0.004	150	93	19	Yes
0.0126	0.007	-	150	9	Yes
0.0156	0.007	-	150	12	Yes
0.0186	0.007	150	137	25	Yes
0.0206	0.007	150	115	19	Yes
0.0126	0.01	-	150	13	Yes
0.0156	0.01	-	150	17	Yes
0.0186	0.01	150	150	18	Yes
0.0206	0.01	150	150	-	No

Table D.5: Back-step (3) from table A.1, with the required number of iterations for each simulation is given for the two CFL numbers in COMSOL and the network prediction. The “-” means the solver diverged.

Back-step (4)					
Max. element size	Inflow velocity	Iterations CFL$_{iter}$	Iterations CFL$_e$	Iterations network	Better?
0.00126	0.01	-	30	-	No
0.00156	0.01	-	26	-	No
0.00186	0.01	150	24	-	No
0.00206	0.01	150	22	-	No
0.00126	0.04	-	149	-	No
0.00156	0.04	-	124	-	No
0.00186	0.04	150	115	-	No
0.00206	0.04	150	93	-	No
0.00126	0.07	-	150	-	Equal
0.00156	0.07	-	150	-	Equal
0.00186	0.07	150	150	28	Yes
0.00206	0.07	24	114	-	No
0.00126	0.1	-	150	-	Equal
0.00156	0.1	-	150	-	Equal
0.00186	0.1	150	150	-	Equal
0.00206	0.1	150	127	-	No

Table D.6: Back-step (3) from table \ref{tab:geom}, with the required number of iterations for each simulation is given for the two CFL numbers in COMSOL and the network prediction. The “-” means the solver diverged.

Back-step (1) with variations						
Max. element size	Inflow velocity	Iterations CFL$_{iter}$	Iterations CFL$_e$	Iterations network - Mirrored	Iterations network - Rotated	Iterations network - Usual
0.0106	0.001	6	15	6	6	6
0.0156	0.001	6	13	6	6	6
0.0206	0.001	6	12	6	6	6
0.0256	0.001	7	11	7	7	6
0.0106	0.004	-	64	100	100	100
0.0156	0.004	29	50	10	10	11
0.0206	0.004	16	37	100	100	12
0.0256	0.004	19	33	14	14	14
0.0106	0.007	-	113	64	64	100
0.0156	0.007	59	57	11	11	11
0.0206	0.007	28	62	17	17	16
0.0256	0.007	33	53	21	21	22
0.0106	0.01	150	150	12	12	100
0.0156	0.01	38	80	17	17	19
0.0206	0.01	40	87	25	25	23
0.0256	0.01	150	150	28	28	24
0.0106	0.012	150	150	-	-	-
0.0156	0.012	150	97	100	100	22
0.0206	0.012	50	105	29	29	28
0.0256	0.012	150	150	100	100	26
0.0106	0.015	150	150	17	17	16
0.0156	0.015	71	131	25	25	23
0.0206	0.015	46	115	37	37	39
0.0256	0.015	150	150	100	100	27

Table D.7: Back-step (1) mirrored in the y-axis and rotated 90 degrees anti-clockwise, with results from network 1. . The “-” means the solver diverged. In the seventh column the results from the “usual” back-step from table A.1 is put as a reference.

Couette flow Outer cylinder radius 0.4 m, inner cylinder radius 0.2					
Max. element size	Inflow velocity	Iterations CFL_{iter}	Iterations CFL_{e}	Iterations Network	Better?
0.014	0.01	55	29	11	Yes
0.016	0.01	54	29	12	Yes
0.018	0.01	51	28	25	Yes
0.02	0.01	50	29	10	Yes
0.022	0.01	47	28	9	Yes
0.014	0.03	150	150	12	Yes
0.016	0.03	54	31	18	Yes
0.018	0.03	53	30	17	Yes
0.02	0.03	49	28	20	Yes
0.022	0.03	48	25	15	Yes
0.014	0.04	61	37	24	Yes
0.016	0.04	55	33	16	Yes
0.018	0.04	52	29	21	Yes
0.02	0.04	51	28	13	Yes
0.022	0.04	48	25	14	Yes
0.014	0.05	57	35	15	Yes
0.016	0.05	62	38	14	Yes
0.018	0.05	53	30	14	Yes
0.02	0.05	51	28	15	Yes
0.022	0.05	47	25	15	Yes
0.014	0.07	57	35	17	Yes
0.016	0.07	55	33	13	Yes
0.018	0.07	53	30	17	Yes
0.02	0.07	51	28	19	Yes
0.022	0.07	47	24	19	Yes
0.014	0.1	57	36	19	Yes
0.016	0.1	55	33	20	Yes
0.018	0.1	52	29	17	Yes
0.02	0.1	50	27	21	Yes
0.022	0.1	47	24	22	Yes

Couette flow Outer cylinder radius 0.08 m, inner cylinder radius 0.04					
Max. element size	Inflow velocity	Iterations CFL_{iter}	Iterations CFL_{e}	Iterations network	Better?
0.0028	0.01	44	18	8	Yes
0.0032	0.01	40	17	11	Yes
0.0036	0.01	40	17	14	Yes
0.004	0.01	40	18	8	Yes
0.0044	0.01	38	17	12	Yes
0.0028	0.03	54	27	11	Yes
0.0032	0.03	52	26	14	Yes
0.0036	0.03	50	26	14	Yes
0.004	0.03	49	25	13	Yes
0.0044	0.03	47	25	13	Yes
0.0028	0.05	55	29	15	Yes
0.0032	0.05	54	29	15	Yes
0.0036	0.05	51	28	15	Yes
0.004	0.05	50	29	17	Yes
0.0044	0.05	47	28	17	Yes

Table D.8: Couette flow with outer cylinder radius 0.4 m and inner cylinder radius 0.2 m, and scaled Couette flow with outer cylinder radius 0.08 and inner cylinder radius 0.04.

Cavity flow								
Dimensions (m)	Max. element size	Wall velocity	Iterations CFL$_{iter}$	Iterations CFL$_e$	Iterations before network	Iterations network	Total iterations	Better?
0.12x0.12	0.00484	0.01	38	23	5	8	13	Yes
0.12x0.12	0.00484	0.08	52	58	14	13	27	Yes
0.12x0.12	0.00484	0.1	52	48	14	10	24	Yes
1.2x1.2	0.0484	0.001	38	23	9	4	13	Yes
1.2x1.2	0.0484	0.008	52	40	9	7	15	Yes
1.2x1.2	0.0484	0.01	52	42	6	11	17	Yes
0.6x0.6	0.00968	0.005	38	23	8	5	13	Yes
0.6x0.6	0.00968	0.05	52	42	10	11	21	Yes
0.6x0.6	0.00968	0.1	51	46	10	17	27	Yes
0.6x0.6	0.00968	0.4	42	43	12	23	35	Yes

Table D.9: Cavity flow with different dimensions, max element size and wall velocities. The iterations before network are performed with pseudo time-stepping with CFL$_e$, then the network is applied which resulted in the total iterations in the 8th column.

Back-step (1), circle with diameter 0.03 m					
x-position (base)	y-position (base)	Iterations CFL$_{iter}$	Iterations CFL$_{e}$	Iterations Network	Better?
0.37	0.045	44	128	20	Yes
0.37	0.05	150	93	22	Yes
0.37	0.055	12	68	11	Equal
0.37	0.06	14	31	13	Equal
0.37	0.065	16	29	12	Yes
0.37	0.07	15	31	12	Yes
0.37	0.075	12	46	12	Equal
0.37	0.08	26	76	13	Yes
0.3	0.04	150	118	12	Yes

Back-step (1), ellipse with a-semiaxis 0.03 m and b-semiaxis 0.02 m					
x-position (base)	y-position (base)	Iterations CFL$_{iter}$	Iterations CFL$_{e}$	Iterations Network	Better?
0.37	0.035	150	90	10	Yes
0.37	0.045	150	89	9	Yes
0.37	0.05	14	111	13	Equal
0.37	0.055	13	100	13	Equal
0.37	0.06	10	58	10	Equal
0.37	0.065	11	40	11	Equal
0.37	0.07	14	25	13	Equal
0.37	0.075	14	23	11	Yes
0.37	0.08	14	41	18	No
0.37	0.09	42	88	13	Yes
0.3	0.04	150	114	10	Yes

Back-step (1), ellipse with a-semiaxis 0.02 m and b-semiaxis 0.03 m					
x-position (base)	y-position (base)	Iterations CFL$_{iter}$	Iterations CFL$_{e}$	Iterations Network	Better?
0.37	0.04	150	129	11	Yes
0.37	0.045	150	123	23	Yes
0.37	0.05	150	94	12	Yes
0.37	0.055	15	66	11	Yes
0.37	0.06	20	38	13	Yes
0.37	0.065	20	31	12	Yes
0.37	0.07	17	30	11	Yes
0.37	0.075	23	46	14	Yes
0.37	0.08	150	75	13	Yes
0.3	0.04	150	122	10	Yes
1.1	0.04	150	100	21	Yes

Table D.10: Back-step (1) from table A.1, with different obstacles. The y-position of the base of the obstacle is mostly adapted, the inflow velocity is 0.008 m/s and the maximum element size is 0.0126. The maximum number of nonlinear iterations for CFL$_{iter}$ and CFL$_{e}$ are set on 150.

Couette, circle with diameter of 0.04 m					
Max. element size	Wall velocity	Iterations CFL$_{iter}$	Iterations CFL$_{e}$	Iterations Network	Better?
0.014	0.01	35	150	10	Yes
0.016	0.01	33	150	17	Yes
0.018	0.01	31	56	11	Yes
0.02	0.01	35	34	9	Yes
0.022	0.01	27	29	10	Yes
0.014	0.04	-	150	-	No
0.016	0.04	46	150	17	Yes
0.018	0.04	41	150	19	Yes
0.02	0.04	49	150	-	No
0.022	0.04	36	145	19	Yes
0.014	0.07	-	150	-	No
0.016	0.07	-	150	100	No
0.018	0.07	44	150	34	Yes
0.02	0.07	43	150	24	Yes
0.022	0.07	46	150	54	No

Couette, ellipse with a-semiaxis 0.03 m and b-semiaxis 0.05 m					
Max. element size	Wall velocity	Iterations CFL$_{iter}$	Iterations CFL$_{e}$	Iterations Network	Better?
0.014	0.01	37	150	10	Yes
0.016	0.01	40	44	19	Yes
0.018	0.01	37	32	13	Yes
0.02	0.01	36	30	12	Yes
0.022	0.01	35	25	9	Yes
0.014	0.04	-	150	13	Yes
0.016	0.04	59	150	20	Yes
0.018	0.04	55	150	22	Yes
0.02	0.04	50	150	24	Yes
0.022	0.04	44	104	23	Yes
0.014	0.07	-	150	44	Yes
0.016	0.07	-	150	61	Yes
0.018	0.07	150	150	40	Yes
0.02	0.07	52	150	25	Yes
0.022	0.07	53	150	30	Yes

Couette, ellipse with a-semiaxis 0.05 m and b-semiaxis 0.03 m					
Max. element size	Wall velocity	Iterations CFL$_{iter}$	Iterations CFL$_{e}$	Iterations Network	Better?
0.014	0.01	34	150	11	Yes
0.016	0.01	33	133	16	Yes
0.018	0.01	35	59	16	Yes
0.02	0.01	31	46	11	Yes
0.022	0.01	27	29	10	Yes
0.014	0.04	43	150	-	No
0.016	0.04	35	150	-	No
0.018	0.04	40	150	30	Yes
0.02	0.04	34	150	-	No
0.022	0.04	32	150	20	Yes
0.014	0.07	40	150	-	No
0.016	0.07	-	150	-	No
0.018	0.07	55	150	-	No
0.02	0.07	46	150	37	Yes
0.022	0.07	37	150	41	No

Table D.11: Couette flow with outer cylinder radius 0.4 m and inner cylinder radius 0.2 m with three different obstacles. The mesh size and velocities are adapted, while the center of the obstacle remains in location x = 0.1 m and y = 0.4 m. The “-” means the solver diverged.

E

Matlab code

Important parts of the code will be discussed here, divided into two parts: code for the interface between Matlab and COMSOL and code for creating unusual structures of the neural network.

Interface between COMSOL and Matlab

LiveLink for Matlab introduces a connection from COMSOL to the Matlab environment [16]. Within this LiveLink, it is possible to set up COMSOL models from Matlab, use Matlab functions in COMSOL, set up model loops and of course analyze the results of the COMSOL simulation [16].

For this thesis, the interface was used to obtain COMSOL simulation results, create data sets and use the neural network output as an input for the COMSOL simulation. First the model is loaded with `mphload('backstep.mph')`, in this case for the back-step model in COMSOL.

To find specific parameters in COMSOL to adapt in Matlab, the COMSOL Model Library is used. With this tool, you are able to search through the COMSOL file to obtain the variable you want to adapt or obtain in Matlab.

Obtain COMSOL simulation results

In order to use data from the COMSOL simulation, you first have to obtain that information in Matlab. There are several functions to obtain result data, such as flow velocities, pressure and the residuals. But also the results of the sensitivity analysis should be obtained, in order to compute the gradients in section 4.1.

There are two methods that are used in this research and both will be discussed. These methods are `mpheval` and `mphgetu`. We will first discuss `mpheval`.

The function `mpheval` is used to obtain solutions and defined variables from the COMSOL simulation. The variables obtained are u, v, p and their residuals and the Reynolds number. The code to obtain the solution vectors is given as follows:

```
outcome = mpheval(model,{'u','v','p'},'refine',1,'dataset','dset10');
```

Besides the values you want to obtain from the simulation, it is also possible to state from which data set the solutions must be obtained. `outcome` also gives information of the coordinates of each point in the solution vector.

Secondly, `mphgetu` is used to obtain the solutions of the sensitivity analysis. A drawback of this function it that it returns one large vector instead of multiple vectors that represent each of the three variables of the sensitivity analysis. That means you need to separate the vector afterwards. Luckily, the vector is ordered logically and it is not a big deal to separate the vector. The code is given as follows:

```
outcome2 = mphgetu(model,'type','fsens','soltag','sol9');
```

Again, it is possible to state from which solution you want to obtain the results with the use of `'soltag'`. Also, `outcome2` does not have any information of the coordinates of each solution point in the vector. In order to obtain the right coordinates, you must use `mphxmeshinfo`:

```
info = mphxmeshinfo(model);
coords = info.nodes.coords;
```

These coordinates are not ordered the same as the coordinates obtained with `mpheval`, therefore a permutation vector is created to effectively shift the values into the same coordinates.

Change settings in COMSOL

During the training of the neural network and data creation, settings for the COMSOL model must be adapted. During the training we want to turn the CFL value to cfl, so we have to refresh the interpolation function, activate manual CFL value and set it to cfl.

```
model.func('int1').active(true);
model.component('comp1').func('int1').refresh;
model.physics('spf').prop('AdvancedSettingProperty').set('CFLNumbExpr',
  ↪  'Manual');
model.physics('spf').prop('AdvancedSettingProperty').set('locCFL', 'cfl');
```

When a data set is created, the CFL number is set to manual and the maximum number of iterations is set to a certain value n.

```
model.physics('spf').prop('AdvancedSettingProperty').set('CFLNumbExpr',
  ↪  'Automatic');
model.sol('sol8').feature('s1').feature('fc1').set('niter', num2str(j));
```

For finding out which Matlab operation should be used, the COMSOL Model Navigator app is used. This app navigates through the loaded COMSOL simulation.

Create data sets

As mentioned in section 4.3, there are 22 network input values for the part of the network that predicts the CFL number, and there are 2 network inputs that pass information of the coordinates to the regression output layer. In order to obtain this information, the first study step should be run. But first, general information independently of the solution is determined, such as the element edge lengths.

```
model = mphload('backstep.mph');

info = mphxmeshinfo(model);
coords = info.nodes.coords;

for i = 1:size(info.elements.tri.nodes,2)
    edge_length1(i) = norm( [coords(1,mesh_info(2,i))
  ↪  coords(2,mesh_info(2,i))]-[coords(1,mesh_info(3,i))
  ↪  coords(2,mesh_info(3,i))] );
    edge_length2(i) = norm( [coords(1,mesh_info(2,i))
  ↪  coords(2,mesh_info(2,i))]-[coords(1,mesh_info(4,i))
  ↪  coords(2,mesh_info(4,i))] );
    edge_length3(i) = norm( [coords(1,mesh_info(4,i))
  ↪  coords(2,mesh_info(4,i))]-[coords(1,mesh_info(3,i))
  ↪  coords(2,mesh_info(3,i))] );
end
```

Then, for a given number of iteration n, the first study step is run. The results of this study step must be obtained for both the element vertices and the element centroids. The residuals are only required for the element vertices.

```
model.sol('sol8').feature('s1').feature('fc1').set('damp', '1');
model.physics('spf').prop('AdvancedSettingProperty').set('CFLNumbExpr',
 ↪ 'Automatic');
model.sol('sol8').feature('s1').feature('fc1').set('niter', num2str(n));
model.sol('sol8').runAll;

data = mpheval(model,{'u','v','p','spf.res_u','spf.res_v',...
    'spf.res_p','spf.cellRe'},'refine',1,'dataset','dset10');
```

All the required data for the solution are put into a table and this table is then written to an Excel file.

```
writetable(data_table,'LocalData.xlsx','Sheet',n);
```

Neural network structures

For the network that uses the RMSE loss in (4.1), creating the network is straightforward when the Matlab manual on neural networks is used. For the network that uses the error estimate loss, however, the network structure is different than usual networks. Creating this network required a special layer and a concatenation layer. The structure of this network is discussed.

First, the network is created with the following code. `NumNeurons` and `NumLayers` are respectively the number of neurons in the hidden layers and the number of hidden layers. We see that a concatenation layer connects the CFL prediction with the second input layer and that the information is forwarded to the custom regression layer.

```
layer = layerGraph;
NumNeurons = Neurons;
NumLayers  = Layers;
    reluname = '';
    layer1 = [
        featureInputLayer(22,'Normalization','none','Name','input')
        batchNormalizationLayer('Name','BN1')
        fullyConnectedLayer(NumNeurons,'Name','fc_begin')
        reluLayer('Name','Relu1')];%activation function
    layer = addLayers(layer,layer1);
    for i = 1:NumLayers
        reluname2 = reluname;
        reluname = append('Relu',num2str(i+1));
        layername = append('fc_',num2str(i));
        if reluname == "Relu_"+num2str(i)+num2str(DropOut1) || reluname ==
 ↪  "Relu_"+num2str(i)+num2str(DropOut2)
            layer2 =
 ↪  [dropoutLayer('Name',append(append('Dropout_',num2str(j)),num2str(i)))
                fullyConnectedLayer(NumNeurons,'Name',layername)
                reluLayer('Name' ,reluname)];
            layer = addLayers(layer,layer2);
            layer = connectLay-
 ↪  ers(layer,reluname2,append(append('Dropout_',num2str(j)),num2str(i)));
        else
            layer2 = [fullyConnectedLayer(NumNeurons,'Name',layername)
                reluLayer('Name' ,reluname)];
            layer = addLayers(layer,layer2);
            if i>1
                layer = connectLayers(layer, reluname2, layername);
            else
                layer = connectLayers(layer, 'Relu1', layername);
            end
        end
    end

    layer5  = [fullyConnectedLayer(1,'Name','reg_end')];
    input2 = featureInputLayer(2,'Normalization','none','Name','input2');
    concat  =    concatenationLayer(1,2,'Name','concat');
    reglayer=    myRegressionLayer2(tbl,2,miniBatchSize,perm_mat,'last');
    layer = addLayers(layer,layer5);
    layer = addLayers(layer,concat);
    layer = addLayers(layer,reglayer);
    layer = addLayers(layer,input2);
    layer = connectLayers(layer,reluname,'reg_end');
    layer = connectLayers(layer,'reg_end','concat/in1');
    layer = connectLayers(layer,'input2','concat/in2');
    layer = connectLayers(layer,'concat','last');
dlnet = layer;
```

The custom regression layer is given below. Here, a template from Matlab [6] is used to create this layer. In this layer, the loss and its gradients are computed.

```
classdef myRegressionLayer2 < nnet.layer.RegressionLayer

    properties
        % (Optional) Layer properties.
        tbl
        iter
        NumInputs
        perm_mat
    end

    methods
        function layer = myRegressionLayer2(table,
    ↪   Iter,numInputs,Perm_mat,name)
            % (Optional) Create a myRegressionLayer.
            layer.tbl = table;
            layer.iter = Iter;
            layer.Name = name;
            layer.NumInputs = numInputs;
            layer.perm_mat = Perm_mat;
        end

        function loss = forwardLoss(layer, varargin)
            X = varargin{1};
            Coords = X(2:3,:);
            Y = X(1,:);
            loss = comp_loss(Y,Coords);
            if isa(Y,'single')
                loss = single(loss);
            else
                loss = double(loss);
            end

        end

        function dLdY = backwardLoss(layer, varargin)
            % (Optional) Backward propagate the derivative of the loss
            % function.
            %
            % Inputs:
            %         layer - Output layer
            %         Y     - Predictions made by network
            %         T     - Training targets
            %
            % Output:
            %         dLdY  - Derivative of the loss with respect to the
            %                 predictions Y

            % Layer backward loss function goes here.
            X = varargin{1};
            Coords = X(2:3,:);
            Y = X(1,:);

            dLdY = gradient_cfl3(layer.tbl,layer.iter,Y,Coords);
            if isa(Y,'single')
                dLdY = single(dLdY);
            else
                dLdY = double(dLdY);
            end
        end
    end
end
```

The loss is computed in the function `comp_loss.m`:

```
function [loss] = comp_loss(Y,Coords)

    model = mphload('test3.mph');
    Cx = double(Coords(1,:))';
    Cy = double(Coords(2,:))';

    COMSOL(model)

    outcome = mphe-
  ↪ val(model,{'spf.res_u','spf.res_v','spf.res_p'},'refine',1,'dataset','dset12');

    loss_u = griddata(outcome.p(1,:),outcome.p(2,:),outcome.d1,Cx,Cy);
    loss_v = griddata(outcome.p(1,:),outcome.p(2,:),outcome.d2,Cx,Cy);
    loss_p = griddata(outcome.p(1,:),outcome.p(2,:),outcome.d3,Cx,Cy);

    loss = sqrt(sum(loss_u)^2 + sum(loss_v)^2 + sum(loss_p)^2);

    for i = 1:length(Y)
        if Y(i) == 0
            loss = loss + 3000;
        end
    end

    if isa(Y,'single')
        loss = single(loss);
    else
        loss = double(loss);
    end

end
```

And the gradients are computed in `gradient_cfl3.m`:

```matlab
function [grad_tot] = gradient_cfl3(tbl,iter,Y,Coords)

    u = double(table2array(tbl(:,25)))';
    v = double(table2array(tbl(:,26)))';
    p = double(table2array(tbl(:,27)))';
    x = double(table2array(tbl(:,23)))';
    y = double(table2array(tbl(:,24)))';
    model = mphload('test3.mph');

    cfl_num = CFL(iter);
    cfl  = ones(size(tbl,1),1)*cfl_num;
    Cx = double(Coords(1,:));
    Cy = double(Coords(2,:));

    %----Write the CFL number to the right coordinates
    for i = 1:length(x)
        for j = 1:length(Cx)
            if single(x(i)) == single(Cx(j)) && single(y(i)) ==
    ↪   single(Cy(j))
                cfl(i)  = Y(1,j);
                if Y(1,j) == 0
                    cfl(i)  = cfl(i) + 0.0001;
                end
                break
            end
        end
    end

    %----Write the CFL number to the .txt file
    fid = fopen('datagrid.txt','wt');
    fprintf(fid,'%%y z T\n');
            for j = 1:length(cfl)
                fprintf(fid,'%f %f %f\n', x(j), y(j),cfl(j));
            end
    fclose(fid);

    model.func('int1').active(true);
    model.component('comp1').func('int1').refresh;
    model.sol('sol9').runAll;

    outcome2  = mphgetu(model,'type','fsens','soltag','sol9');
    outcome2  = outcome2(~isnan(outcome2));
    sens_u  = zeros(2025,1);
    sens_v = sens_u;
    sens_p = sens_u;
    i  = 1;
    k  =1;
    while i < length(outcome2)
        sens_u(k)  = outcome2(i);
        sens_v(k)  = outcome2(i+1);
        sens_p(k)  = outcome2(i+2);
        i  = i + 3;
        k  = k +1;
    end

    info = mphxmeshinfo(model);
    coords = info.nodes.coords;

    sens_u = griddata(double(single(coords(1,:))),double(single(coords(2,:))),...
        sens_u',double(Cx),double(Cy));
    sens_v = griddata(double(single(coords(1,:))),double(single(coords(2,:))),...
        sens_v',double(Cx),double(Cy));
    sens_p = griddata(double(single(coords(1,:))),double(single(coords(2,:))),...
        sens_p',double(Cx),double(Cy));

    outcome = mpheval(model,{'spf.res_u','spf.res_v','spf.res_p'},'refine',1,'dataset','dset12');

    res_u = griddata(outcome.p(1,:),outcome.p(2,:),outcome.d1,Cx,Cy);
    res_v = griddata(outcome.p(1,:),outcome.p(2,:),outcome.d2,Cx,Cy);
    res_p = griddata(outcome.p(1,:),outcome.p(2,:),outcome.d3,Cx,Cy);

    sens_u = sens_u.*res_u;
    sens_v = sens_v.*res_v;
    sens_p = sens_p.*res_p;

    outcome3 = mpheval(model,{'u','v','p'},'refine',1,'dataset','dset12');
    u_new = griddata(outcome3.p(1,:),outcome3.p(2,:),outcome3.d1,x,y);
    v_new = griddata(outcome3.p(1,:),outcome3.p(2,:),outcome3.d2,x,y);
    p_new = griddata(outcome3.p(1,:),outcome3.p(2,:),outcome3.d3,x,y);
    du = -(u_new - u)./cfl';
    dv = -(v_new - v)./cfl';
    dp = -(p_new - p)./cfl';

    x = single(x);
    y = single(y);
    x = double(x);
    y = double(y);

    grad1 = griddata(x,y,du,Cx,Cy);
    grad2 = griddata(x,y,dv,Cx,Cy);
    grad3 = griddata(x,y,dp,Cx,Cy);

    grad_tot = mphtable(model,'tbl2').data(1)*(grad1.*sens_u+grad2.*sens_v+grad3.*sens_p);

    if isa(Y,'single')
        grad_tot = single(grad_tot);
    else
        grad_tot = double(grad_tot);
    end

    grad_tot = ones(3,size(grad_tot,2)).*grad_tot;
    disp(['The CFL numbers are ', num2str(Y)])

end
```