Output Error Estimation for Unsteady Flows Using Reconstructed Solutions

Effect of Compression and Reconstruction of Unsteady CFD Data using Neural Networks and PODs on Error Estimates

Arnish Sitaram

Delft University of Technology

TU Delft

Output Error Estimation for Unsteady Flows Using Reconstructed Solutions

Effect of Compression and Reconstruction of Unsteady CFD Data using Neural Networks and PODs on Error Estimates

by

Arnish Sitaram

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday September 25, 2023 at 2:00 PM.

Student number:	4645200	
Project duration:	November 1, 2022 – August 30, 2023	
Thesis committee:	Dr. R. P. Dwight,	TU Delft, Committee chair
	Dr. S. J. Hulshoff,	TU Delft, Thesis Supervisor
	Dr. C. C. De Visser,	TU Delft, External member
	Ir. T. P. Hunter,	TU Delft, Daily Supervisor

If somebody else can do it,
you can do it too.

R.K. Sitaram

Preface

Dear reader,

This thesis started with an interest in machine learning and artificial intelligence, which resulted in a gap in knowledge that I would like to have filled in during my short academic career. I expressed my wishes during a research proposal meeting with Dr. Steven Hulshoff, which resulted in this thesis. Together with the daily supervision of Ir. Thomas P. Hunter and the global supervision of Dr. Steven Hulshoff, this thesis has been an instructive and unique experience that will mark an end to my academic career.

I would like to thank Dr. Steven Hulshoff and MSc Thomas P. Hunter for their continuous supervision and interesting discussions which elevated this thesis to another level. I am grateful for the time we spent together brainstorming and bouncing ideas back and forth. As the end of my academic career comes closer, I look forward to starting my job at ABN Amro in Amsterdam.

My entire academic career, I have been supported by my family and friends. I would not be in the place where I am right now without their continuous support and everlasting love. I would like to thank you all from the bottom of my heart for shaping me into the person I am today. For all the memories we share and cherish, the good times we had, and the bad times we overcame. I am grateful to be surrounded by you all during this phase in my life.

Arnish Sitaram
Delft, August 2023

Abstract

Unsteady numerical simulation has been proven to be an essential tool for research. The quality of the results can be improved by using mesh adaptation. Mesh adaptation uses error indicators to refine the mesh in regions with high errors. The error indicators used are output errors with the most accurate output error estimation method being adjoint-based error estimation. However, for this method, the primal solution needs to be stored, which is storage intensive, especially for large unsteady simulations. The method proposed in this thesis uses a neural network autoencoder to compress and reconstruct the primal solution. This solution is compared to a reconstructed solution using Proper Orthogonal Decomposition (POD).

The one-dimensional unsteady Burgers equation is used as validation for the methods using a manufactured solution while the lid-driven cavity flow is investigated using the proposed method. The manufactured solution of the one-dimensional burgers case could be exactly reconstructed using two POD modes. For the autoencoder a small latent space was used. For low resolutions, the small latent space did not prove to be a problem as the primal and residual could be captured accurately. However, for higher resolutions, the reconstruction error of the autoencoder became dominant for the residuals and resulted in erroneous adjoint-based error estimates while the primal remained qualitatively similar.

For the lid-driven cavity flow, the POD was still able to capture the solution using a low number of modes due to the smoothness of the solution. This resulted in an unfair comparison between the POD and autoencoder reconstructed solutions. The reconstructed autoencoder error estimates for lower resolutions were more accurate due to the latent space being large enough to capture the residual of the discrete primal accurately enough. When moving to higher resolutions, the autoencoder was not able to reconstruct the residual accurately enough leading to erroneous error estimates. Therefore, the latent space of autoencoders should be sufficiently large in order to gain an accurate reconstruction of the residual. If the latent space is large enough, the error estimate is accurate and the local error estimates can be used as a first iteration error indicator for mesh refinement.

Contents

List of Figures

List of Tables

List of Abbreviations

AMR	Adaptive Mesh Refinement
ANN	Artificial Neural Network
BC	Boundary Condition
CFD	Computational Fluid Dynamics
DMD	Dynamic Mode Decomposition
DNS	Direct Numerical Simulation
EOA	Enhanced Online Algorithm
GAN	Generative Adversarial Network
HPC	High Performance Computing
IC	Initial Condition
LES	Large Eddy Simulation
MSE	Mean Squared Error
NN	Neural Network
PDE	Partial Differential Equation
POD	Proper Orthogonal Decomposition
QoI	Quantity of Interest
ROM	Reduced Order Model
ROR	Reduced Order Representation
SVD	Singular Value Decomposition

List of Symbols

Sign	Description
$(\cdot)'$	Fluctuation of quatity $(\cdot)$
$(\cdot)_H$	Quantity defined on the coarse space
$(\cdot)_{\textbf{AE}}$	Quantity based on the autoencoder reconstructed solution
$(\cdot)_{\textbf{MMS}}$	Quantity based on the manufactured solution
$(\cdot)_{\textbf{POD}}$	Quantity based on the POD reconstructed solution
$(\cdot)_h$	Quantity defined on the fine space
CR	Compression ratio
I	Time domain
L^*	Adjoint operator
L	Linear operator
N	Spatial resolution
Re	Reynolds Number
T	Final time
Δt	Time step
Ω	Continuous domain
Ψ	Continuous adjoint
$\bar{J}$	Quantity of Interest
δJ	Error
$\delta \bar{J}_{est}$	Adjoint-based error estimate
$\delta \mathbf{u}$	Infinitesimally small perturbation of the solution
$\mathbf{R}$	Residual
$\mathbf{u}_h^H$	Coarse solution injected into the fine space using spatial interpolation
$\mathbf{u}_0$	Solution which is used for linearisatioin
$\mathbf{u}$	Unknown solution
ν	Viscosity coefficient
$\partial\Omega$	Boundary of domain Ω
ε_{est}	Local error estimates based on the adjoint-weighted residual
f	Source function
n_{var}	Number of variables
r	Number of modes

1 Introduction

1.1. Background and motivation

Computational Fluid Dynamics (CFD) has become an essential tool in the investigation of fluid flows [1]. The increase in computational availability of numerical simulations in recent decades has permitted more complex fluid flow models to be simulated [2]. This would produce large amounts of computational data especially if the flow problem is of unsteady nature. As the improvement in computational capacity has been remarkable, the storage capacity has not been able to keep up. This leads to a performance gap between CPU and storage which is getting wider [3]. Multiple parties such as NASA [4], research agencies [5, 6] and corporations, have indicated the need for more precise and complicated fluid dynamic simulations. This need is satisfied by constructing effective and accurate meshes. However, creating such meshes necessitates careful surveillance and heavily relies on the user's expertise. Besides this, the use of CFD is computationally expensive and storage intensive. These disadvantages limit the broad usage of CFD.

A way to alleviate the computational and storage burden while increasing the precision of the obtained solution is to make use of adjoint-based mesh adaptation [7]. Adjoint-based mesh adaptation is a procedure in which local error estimation is used to identify cells that need refinement to achieve a more accurate solution. This means that adjoint-based mesh adaptation can initially start by computing the solution on a very coarse mesh and then iterating the mesh so that refinement is added in the parts where it is needed. Especially for unsteady problems, where the computational costs and storage burden are excessively high, adjoint-based mesh adaptation offers a viable alternative.

Mesh adaptation uses an indicator to identify cells within the mesh that need refinement. A commonly used indicator is the output error. The output error is the error of a Quantity of Interest (QoI) which is used to identify the local contributions of each cell. The output error is more viable to be used as an error indicator compared to discretisation error and residual error estimates [8]. Determining what parts of the primal can be compressed and how much the primal solution can be compressed without losing accuracy can be interpreted as an optimisation problem.

The most accurate way to identify these cells is by making use of adjoint-based error estimation [9]. This method computes local error estimates based on the residual and the adjoint solution [10]. This method has two problems. The adjoint solution is obtained by solving an adjoint equation that uses the primal solution as input. Solving the adjoint equation imposes an additional computational burden. Next to this, the residual also needs the primal solution as an input. For unsteady problems, this means that the primal solution needs to be stored at every timestep, which is very storage intensive. One option would be to store the primal solution on disk. However, this makes the computation of the adjoint-based error estimates very slow.

Numerous methods may be used to get over a numerical method's limitations in terms of computational cost and storage needs. One of these methods is to use compression techniques to reduce the storage footprint. Techniques such as Proper Orthogonal Decomposition (POD) [11], machine learning techniques [2, 12] or a combination of both [13–15] have been used to compress primal data of fluid flow solutions to lessen the storage burden imposed by CFD simulations.

By applying these compression techniques to the primal solution, the resulting compressed flow will be different from the inputted primal flow. To compare both primal solutions, the compressed primal needs to be decompressed or reconstructed. The error introduced by the reconstruction can then be studied using error estimation techniques. The goal is to obtain the most accurate compression and reconstruction without the loss of useful information. This will result in efficient storage of the primal.

1.2. Output error estimation

Several approaches have been developed to determine the output error. The most accurate means of determining the output error is adjoint-based error estimation [16]. This thesis focuses on this method to gain insight into the output errors. This method depends on the solution of the adjoint problem, which describes a sensitivity problem related to the fluid equations. In literature, the adjoint problem is sometimes referred to as the dual problem while the fluid problem is referred to as the primal problem. Even though adjoint-based error estimates have excellent accuracy and a solid mathematical foundation combined with the linear nature of the adjoint problem, the associated computing and storage costs are still too high for broad application.

The computational and storage capacity needed for using the adjoint-based error estimation method subsides from the cost of solving the adjoint problem by making use of the primal solution which is stored at each time step.

To come up with an error estimate, the adjoint needs to be solved on a finer discretisation than the primal problem. The primal solution will be interpolated onto this fine space to compute the error estimates. While this method lessens the computational burden of the fine primal, it also adds inaccuracies to the error estimates.

Similar to how the primal solution depends on its preceding time step, the adjoint problem needs to be solved backwards in time as it represents a sensitivity problem. Since the adjoint solutions use the primal solutions to come up with an adjoint solution, the primal solutions at all time steps must be stored and callable. Therefore, not only do the adjoint solutions need to be stored during the backwards loop, but also the primal solution during the forward loop needs to be stored, which results in the large storage capacity needed for applying adjoint-based error estimation.

1.3. Storage optimisation of output error estimation methods

The impact of the two main issues, identified in Section 1.1, can be reduced by improving the output error estimation process for optimal usage of computational and storage overhead. Several methods have been proposed to achieve this optimisation. One of these methods is to make use of Artificial Neural Networks (ANN) to compress and reconstruct the primal solution [2, 12, 13]. This enables the primal to be stored while taking up less storage overhead. Additionally, when the stored primal is called, it needs to be reconstructed to a solution resembling the original state. This means that perfect reconstruction is not likely and will introduce errors in the reconstructed solution.

Another method to lessen the storage needs of the adjoint-based error estimation method is to use Reduced Order Models (ROM) and in particular POD [17, 18]. PODs represent the primal solution as a simplified mathematical representation that captures its essential dynamics while significantly reducing the computational complexity. This will reduce the computational and storage overhead needed for the primal. However, the primal still needs to be reconstructed from this simplified representation, which introduces errors in the reconstructed primal solution.

1.4. Thesis need, scope and outline

The need for cheap and accurate compression and reconstruction algorithms has been identified above. In combination with the trend to apply machine learning, compression and reconstruction to gain accurate adjoint-based error estimates can be achieved such that the storage intensity is lessened. Many of the proposed approaches only compressed and reconstructed the primal solution but did not study the impact on the adjoint-based error estimates and did not map out the inaccuracy introduced by the compression and reconstruction.

Considering the identified need, the trends seen, and the broad application of machine learning in various sectors, it was decided that the influence of compression and reconstruction on the adjoint-based error estimate should be mapped out. Therefore, the following main research question and research objective were established.

Research Question

What is the effect of compression
and reconstruction of unsteady CFD data on adjoint-based output error estimation?

Research Objective

Compress and reconstruct unsteady CFD data for accurate adjoint-based output error estimation.

Secondary research questions can be established using the research question mentioned above. These secondary questions, as stated below, aid in hypothesis testing and create the basis for the work done in this thesis.

1. What are the state-of-the-art POD and neural network applications for compression and reconstruction in a CFD context?
2. What is the state-of-the-art adjoint-based error estimation methods in a CFD context?
3. How can a primal compression and reconstruction neural network and POD be developed and evaluated in a CFD context?
4. How much error is introduced by POD and neural network compression and reconstruction?

The first question ensures the current neural network application for compression and reconstruction within a CFD context is investigated. This can be done through the examples given in literature [2, 13, 17, 19–21]. The first question enables a comparison between the different methods used and the implications they would have on compression and reconstruction of the primal solution. Based on this, a decision is made about how to develop a specific compression and reconstruction algorithm for accurate error estimation.

The second question establishes a reference for the current error estimation methods such as [10, 17] but also [22–26]. These papers will provide a framework for the error estimation method that will be used to map out the errors introduced by compression and reconstruction. Additionally, insights for adjoint-based error estimation within the CFD context have been documented as well in these papers.

The compression and reconstruction neural network's capability is then evaluated in the third question. Gaining a better grasp of the neural network's design and training process allows one to make more educated decisions to maximize the neural network's potential to fulfil the primary research aim.

Finally, the last question discusses how the suggested technique is integrated into the output error estimation method. The integration must be examined in order to present a framework that is practical and has the ability to solve the thesis need from which the research objective arises.

The goal of this thesis is to provide meaningful examples of the capacity of the suggested method. Consequently, one-dimensional and two-dimensional test cases were solved during this research. The hypothesis tests will revolve around the output error introduced by ANNs.

This thesis will explore the ability of neural networks to compress and reconstruct data. The compression and reconstruction will be done on unsteady CFD data. Using an adjoint-based error estimation method, error estimates will be generated to map out the effect of the autoencoder neural network and POD on the error estimates.

The thesis is structured in the following way. Chapter 2 presents a short review of the concept needed to answer the main research question and corresponding sub-questions. Chapter 3 explains the method proposed to achieve compression and reconstruction of the primal. In this chapter, the error estimation framework is also explained. After this, in Chapter 4, the proposed method is applied to the unsteady one-dimensional Burgers equation, which serves as a validation case. Next, the proposed method is applied to the lid-driven cavity flow problem in Chapter 5 which serves as the main test case of this thesis. Finally, closing off the thesis, the conclusion and future recommendation are given in Chapter 6.

2 Thesis Background

With the evolution of multi-core processing units into many-core units and the rapid growth in the number of cores in supercomputers [27], the solution of more complex flow cases could be studied. These advances in computational processing power in combination with the lower computing cost led to a larger gap between processing power and storage due to bandwidth constraints and storage-access times given by high latency input/output operations [12]. In a CFD context, when performing a large CFD simulation where every time step is several gigabytes in storage, this gap results in a storage bottleneck where high-performance computing (HPC) machines are constrained by the need to save, visualize or analyze data [12, 20, 27].

To achieve high-fidelity solutions, flow cases are simulated by solving the Navier Stokes equations directly, also known as Direct Numerical Simulation (DNS). The computational cost of DNS scales with Re^3. Large Eddy Simulation (LES), which simulates large scales and models the small scales, is less costly but sacrifices accuracy and access to small-scale flow data. Both DNS and LES have high computing and storage needs.

Additionally, it is important to take into account the curse of dimensionality, particularly in two- and three-dimensional contexts. The difficulties and complications that develop when dealing with high-dimensional problems are referred to as the ”curse of dimensionality,” which are particularly relevant in CFD when trying to address issues involving an extensive number of variables or dimensions. The computing cost and complexity of addressing the problem rise exponentially as the number of variables rises.

In CFD, a number of techniques are used to relieve the storage issue and the curse of dimensionality. Dimensionality reduction, adaptive meshing, and parallel computing are some of these techniques. Likewise, with the development of accurate and effective data compression, the aforementioned problems can be resolved as well.

2.1. CFD compression and reconstruction using Neural Networks

Machine learning with an emphasis on Neural Networks NNs, was used in several different ways to compress the primal solution. This was done to alleviate the storage burden CFD has, especially for problems of unsteady nature. The NN will compress the primal solution such that a low-resolution primal is achieved. During the compression, the NN should be able to distinguish which regions of the solution can be compressed and which regions cannot. This should enable the NN to maintain important characteristics of the high-resolution primal while constructing the low-resolution primal.

Once the low-resolution primal is constructed, another NN is used to reconstruct a high-resolution primal for the adjoint problem. This reconstructed high-resolution primal should be accurate enough to solve the adjoint problem backwards in time. This will yield the adjoints which is used together with the primal residual to estimate the local error of each element. The adjoint problem is storage intensive as it requires the primal solution to be available at every time step. This means that the primal solution needs to be saved on hard drives. By converting the primal to a lower resolution, storage can be saved and the computation can be sped up.

A method for Adaptive Mesh Refinement (AMR) for LES using reduced-order primal solutions can be seen in [17]. The method uses POD to create a Reduced-Order Representation (ROR) of the primal flow solutions, which reduces the storage requirements for LES. The proper orthogonal decomposition-based ROR and the Enhanced Online Algorithm (EOA) based on incremental Singular Value Decomposition (SVD) are successful in building the ROR online, making adjoint-based AMR feasible for practical applications. The adjoint-based error estimation procedure introduced is verified using the method of a manufactured solution. The ROR-driven AMR strategy is studied using a 1D unsteady Burgers problem with a multi-frequency forcing term, and the numerical results demonstrate the effectiveness of the proposed method. This provides a promising approach for reducing the computational cost of LES simulations while maintaining accuracy.

A deep learning approach to in-situ data compression of large turbulent flow simulations is proposed in [19]. The proposed fully convolutional autoencoder architecture compresses turbulent flow snapshots by a factor of 64 with a single pass and allows for arbitrarily sized input fields. The autoencoder outperforms a randomized single-pass SVD with a similar compression ratio and yields comparable performance to a higher-rank decomposition. This is achieved with an order of magnitude less compression in terms of preserving important statistical quantities such as turbulent kinetic energy, enstrophy, and Reynolds stresses.

Another deep learning approach for compression and deconstruction is using a physics-informed deep learning technique based on vector quantization to generate a discrete, low-dimensional representation of data from simulations of three-dimensional turbulent flows [20]. The accuracy of the model is assessed using statistical, comparison-based similarity and physics-based metrics. The training data set is produced from DNS of an incompressible, statistically stationary, isotropic turbulent flow. The performance of the data compression scheme is evaluated not only with unseen data from the stationary, isotropic turbulent flow, but also with data from decaying isotropic turbulence, a Taylor-Green vortex flow, and a turbulent channel flow. The results show that the model based on vector quantization can offer a high compression ratio of 85 with a Mean Squared Error (MSE) of $\mathcal{O}(10-3)$. This achieves predictions that faithfully reproduce the statistics of the flow, except at the very smallest scales where there is some loss. Compared to the recent study of [19], which was based on a conventional autoencoder, this model improves the compression ratio by more than 30% and reduces the MSE by an order of magnitude of 1. The compression model is an attractive solution for situations where fast, high-quality, and low-overhead encoding and decoding of large data are required.

A new in situ compression method for CFD data using deep learning is presented in [2]. The proposed method uses a generative adversarial network (GAN) to compress and reconstruct CFD data. The method samples small patches from the CFD data and trains the GAN, which includes two convolutional neural networks: the discriminative network and the generative network. The proposed method has advantages in compression time and can adjust the compression ratio according to acceptable reconstruction effect. Experimental results demonstrate the good generalization of the proposed method on many datasets.

In [21], a method for reducing both the computation cost and storage for Lattice Boltzmann flow simulations is presented. The method employs convolutional autoencoders and residual connections to reduce the dimensionality of the data while compressing the solution and learning the dynamics based on this compressed representation.

An innovative way in which data compression is used is seen in [13]. In this paper, the computed results of high-order discontinuous Galerkin computations are compressed using a NN-based autoencoder in combination with POD. This resulted in a compressed state of the solutions. Using a vectorial kernel orthogonal greedy algorithm which uses radial basis functions, the dynamics are learned. After that, the solutions could be recovered at any instance of time from the compressed state.

2.2. Proper Orthogonal Decomposition

Simulating complex physical systems can be computationally straining and storage intensive. To alleviate these constraints, Reduced Order Models (ROMs) can be applied to these systems. The application of ROMs, first introduced for fluid flows in [28], will reduce the computational cost and storage needed for high-fidelity solvers.

There are two types of reduced-order modelling approaches: intrusive and non-intrusive. The difference between these two approaches is that for the intrusive reduced-order modelling approach, access is needed to the complete model and solvers, which are generally unavailable when dealing with governing equations of fluid flows. The non-intrusive reduced-order modelling technique is based on snapshots from high-fidelity tools, such as numerical approximations produced from CFD or wind tunnel observations. The non-intrusive ROMs do not require any knowledge of the underlying equations while still being able to approximate the flow field's leading characteristics. Therefore, the focus will be on non-intrusive ROMs

The non-intrusive ROM approach approximates the full solution is only using a small subset of the data using dimensionality reduction techniques. There are several techniques to reduce dimensions, however, the modal decomposition method is the most used method. Modal decomposition can be used to project the full solution on a reduced-order basis in order to approximate the full solution. The Proper Orthogonal Decomposition (POD) method and the Dynamic Mode Decomposition (DMD) method are the two main modal decomposition techniques. Although each method has advantages of its own, the POD method is seen to be the most prominent [29] and has the greatest potential for the thesis goal.

The fundamental idea behind the POD approach is to decompose a given vector field into orthogonal bases, sometimes referred to as spatial functions, which each capture a component of the flow's total mode energy. It should be emphasized that the spatial functions identified with POD are sorted by total mode energy with the first spatial mode having a higher total mode energy than the second spatial mode. Based on the spatial modes which retain a given threshold of total mode energy (usually 99% [30]), the number of dimensions of a flow field can be reduced. This method will be elaborated on in chapter 3.

2.3. Neural Network Compression and Reconstruction techniques

In Section 2.1, examples are discussed on how to compress and reconstruct CFD data using neural networks. In this section, data compression is discussed as well as data compression using machine learning.

2.3.1. Data compression

In a CFD setting, data compression may be used for data checkpointing as well as post-processing. Data checkpointing means saving compressed data and utilizing it to continue the simulation [19]. However, the data cannot simply be compressed randomly. The compression needs to be done accurately enough such that the simulation is able to restart using the compressed data in memory. The compression needs to ensure the data does not have a significant impact on the simulated flow's long-term behaviour and that the compressed data retains the essential statistical properties of the flow with reasonable accuracy [20].

Depending on how much information is lost during the data compression process, compression techniques can be classified as lossless or lossy. Lossless compression is when data is compressed in such a way that the reconstructed data contains no or machine precision errors. Lossy compression can be characterized by the controllable level of error that is introduced in the data while reconstructing. While lossless compression is able to maintain the accuracy of the data, its compression capabilities are limited and it can be storage intensive [20]. Since the main problem of the proposed thesis will consist of large unsteady simulations that are storage and computationally intensive, lossless compression is less attractive.

Two lossy data compression methods are truncation-based approaches and transformation-based approaches [20]. The floating point accuracy of the data is reduced through truncation-based methods which are comparable to interpolation methods such as nearest neighbor, linear and polynomial splines. Transformation-based approaches encrypt data into a certain representation from which it can be retrieved. The representation usually arranges the data according to some concept of priority. This ordering facilitates lossy data reduction by keeping the relevant components and eliminating the irrelevant ones. This can be seen in POD as well by ordering the modes by total mode energy.

Within lossy classification, there are two techniques that can be interesting to look at which are mesh reduction and derived representation [31]. Mesh reduction techniques reduce the mesh's size, or its number of vertices and cells, which lowers its storage requirements. Mesh reduction differs from lossy compression in that it affects the underlying mesh. By making coarser meshes that mimic the original ones is one approach, a smaller mesh is obtained which has the advantage of requiring less storage to process, improving interactivity [31]. This is not done for lossy compression.

Extreme data reduction is possible with derived representation approaches since they completely discard the source data. Alternative representations are created and kept as an alternative, allowing analysis algorithms to return results that are comparable to those obtained when using the original data.

2.3.2. Machine Learning Data Compression

To achieve lossy data reduction in storage, machine learning techniques could be applied to data. There are two architectures that seem very promising to satisfy the research objective. These two architectures are autoencoders and variational autoencoders. Autoencoders are a special class within encoder-decoder models in which the input and output are the same [32]. The encoder and decoder components of an autoencoder employ multi-layered deep neural networks to reduce the dimensionality of data [13]. The encoder stores the representational information needed to convert the data from its original high-dimensional state to a latent space. The decoder learns how to rebuild from the latent space to the original state. Both the encoder and decoder are tensors that are trained using typical backpropagation and optimization approaches in neural networks [33].

Variational autoencoders build upon the concept of autoencoders. Besides compressing and reconstructing data, variational autoencoders also learn a structured latent space representation [34] from which new samples can be generated. Variational autoencoders are therefore generative models. Variational autoencoders use probabilistic encoders and decoders, enabling them to learn the underlying probability distribution of the data [35]. The encoding network of variational autoencoders will output two vectors instead of one like autoencoders. The latent representation of the input is then drawn from the normal distribution defined by these two vectors. The difference between an autoencoder and a variational autoencoder is seen in Figure 2.1. For the purpose of this research, the probabilistic approach of generating new samples is not needed. Therefore, the use of a variational autoencoder is redundant for the purpose of this thesis.

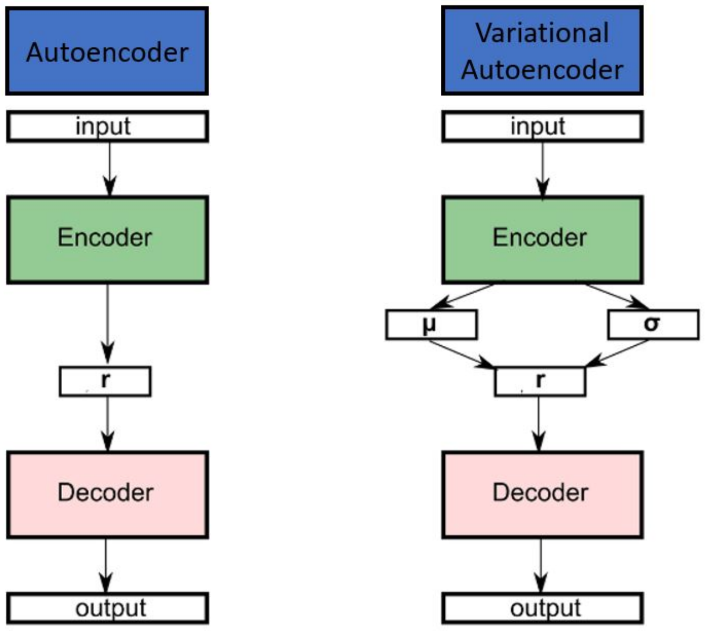

Figure 2.1: Difference between an autoencoder and a variational autoencoder [36]

2.4. Error estimation

There are several approaches that try to estimate the error. The approaches can be categorized into two categories, a priori and a posteriori error estimations. The first approach attempts to anticipate the error before any mesh-based solutions have been identified, depending purely on prior knowledge, mesh topology, and mesh location. This has significant drawbacks and has not been shown to function reliably and robustly across a variety of challenges. However, a priori error estimations have been useful for grid studies. The second approach attempts to estimate the error after a solution has been computed [37]. It considers particular data about the calculated solution, including residuals, gradients, or local error indications [38, 39]. A posteriori estimates offer a more precise evaluation of the inaccuracy in the calculated solution and can direct the optimisation of the numerical solution or AMR. A posteriori error estimates are especially helpful when the exact solution is unknown or difficult to determine. A posteriori estimates are frequently calculated using error indicators, which describe the local error at various points across the computational domain.

The difference between a priori and a posteriori error estimations highlights a clear advantage of using a posteriori error estimates for the purpose of this thesis. The two main approaches to evaluating a posteriori error estimates are metric-based and adjoint-based error estimation.

The first approach, known as metric-based error estimation, drives the error estimation procedure with characteristics from the flow solution. These characteristics are described using measurements that are relevant to the problem. Metric-based error estimation is computationally inexpensive, but it can be inefficient in lowering the output error in terms of a specific QoI, such as lift. The QoI does not always have to be connected with the resolution of local flow characteristics [40, 41].

Furthermore, in order to be successful, the chosen metrics must relate to the problem and the chosen QoI. This necessitates a thorough previous understanding of the problem under consideration. This indicates a lack of generality, which is often solved by using more complicated metrics [42, 43]. These techniques can be effective for highly specific issues, but they are less efficient than goal-oriented mesh adaptation and do not generalize well [44].

Output-based error estimate, on the other hand, is specially designed to decrease the errors in QoI prediction. Output-based error estimates can be accomplished by estimating numerical errors or by solving the variational problem. They were found to be equal to approaches such as the adjoint-weighted residual, but less precise and less computationally costly. If the cost of the adjoint solution is too expensive, these methods are still in use [45, 46].

The second approach, known as adjoint-based error estimation, drives the error estimation procedure by solving the adjoint problem. The adjoint problem is the related variational problem, that may be obtained from the continuous equations or discretised versions of the considered problem. Initially, the adjoint problem is more difficult to set up since it necessitates the formulation of related variational problems, and it is also more computationally costly [40]. However, it is more accurate, requiring coarser meshes and, as a result, a lower computing cost since it can converge to a QoI value more quickly. This approach, which will be a major emphasis of this thesis, allows for effective prediction of output error and localization of error sources. Chapter 3 delves deeper into output error prediction approaches, highlighting their fundamental limitations and arguing the need to design tools that decrease computational overhead and associated storage needs.

3 Methodology

This chapter explains the approach that was taken to achieve the thesis objective and answer the main research question. Both compression methods discussed in Chapter 2, as well as the output error estimation method, are explained. First, POD compression and reconstruction is explained in Section 3.1. After this, neural network compression and reconstruction will be discussed in Section 3.2. Next, output-based error estimation is elaborated upon in Section 3.3. Finally, the output-based error estimation framework is combined with the results from the compression techniques yielding the proposed method in Section 3.4.

To obtain primal data, the finite element method is used for solving the Partial Differential Equations (PDEs) numerically. To evaluate the error estimation ability, this thesis focused on two sets of PDEs: the 1D Burgers equation and the Navier Stokes equations for the lid-driven cavity flow. The 1D Burgers equation will be used as validation for the proposed method while the lid-driven cavity flow is used as the main test case. These PDEs will be investigated using different spatial resolutions and will employ Crank-Nicolson time stepping schemes for additional stability. For the remainder of the chapter, the primal solution can be assumed to be given.

3.1. POD Compression and Reconstruction

POD was used to compress and reconstruct the discrete primal solution. For the POD, data that is a function of both space and time $\mathbf{y}(x,t)$ is inputted. $\mathbf{y}$ could represent the total flow solution or the fluctuations of the flow solutions from the decomposition $\mathbf{u}_{sol} = \overline{\mathbf{u}_{sol}} + \mathbf{u}'_{sol}$, depending on the problem. If separation of variables is applied, the function $\mathbf{y}(x,t)$ transforms to:

$$\mathbf{y}(x,t) = \sum_{j=1}^{m} \mathbf{u}_j(x)\mathbf{a}_j(t) \tag{3.1}$$

where $\mathbf{u}_j$ denotes the spatial functions and $\mathbf{a}_j$ denote the coefficients. The goal is to identify orthogonal bases that are used to approximate most of the data using the smallest number of terms. The POD finds this decomposition of bases. In Equation 3.1 the orthogonal bases are represented by spatial modes $\mathbf{u}_j(x)$. If the data at spatial locations $x = x_1, x_2, ..., x_n$ and times $t = t_1, t_2, ..., t_m$ of $\mathbf{y}(x,t)$ are calculated, these can be represented in an $m \times n$ matrix $\mathbf{Y}$ as:

$$\mathbf{Y} = \begin{bmatrix} y(x_1,t_1) & ... & y(x_n,t_1) \\ \vdots & \ddots & \vdots \\ y(x_1,t_m) & ... & y(x_n,t_m) \end{bmatrix} \approx \underset{m\times m}{\mathbf{U}} \; \underset{m\times n}{\Sigma} \; \underset{n\times n}{\mathbf{V}^T} \tag{3.2}$$

In Equation 3.2, $\mathbf{U}$ is of size $m \times m$ and contains all the spatial functions. Σ is a matrix of size $m \times n$ containing the non-negative singular values ordered by size on its main diagonal and $\mathbf{V}^T$ is a matrix of size $n \times n$ containing the time-dependent coefficients. Note that $\Sigma \mathbf{V}^T$ is equal to $\mathbf{a}_j(t)$ and $\mathbf{U}$ is equal to $\mathbf{u}_j(x)$ seen in Equation 3.1.

Since Σ is ordered such that the largest singular value is located first on the main diagonal, small values can be omitted, introducing a truncation to the matrices. This truncation can be seen in Equation 3.3.

$$\mathbf{Y}=\begin{bmatrix} y(x_1,t_1) & ... & y(x_n,t_1) \\ \vdots & \ddots & \vdots \\ y(x_1,t_m) & ... & y(x_n,t_m) \end{bmatrix} \approx \underset{m\times r}{\mathbf{U}_r} \quad \underset{r\times r}{\mathbf{\Sigma}_r} \quad \underset{r\times n}{\mathbf{V}_r^T} = \mathbf{Y}_r \tag{3.3}$$

The number of modes r is determined according to a total mode energy criterion which is dependent on the data. The criterion can be calculated as follows:

$$\text{Total mode energy} = \frac{\sum \mathbf{\Sigma}_r^2}{\sum \mathbf{\Sigma}^2}$$

Starting from the lowest number of modes of $r = 1$, the number of modes needed to satisfy the criterion is determined iteratively by increasing the number of modes and recalculating the total mode energy.

When applying this technique to a velocity vector, the velocity data first needs to be converted from finite elements to points. The data in points can then be inputted into the POD as **Y** to obtain $\mathbf{Y}_r$ which is equal to $\mathbf{u}_{POD}$. When the POD is applied on the fluctuations of the flow, the mean flow should be added to obtain $\mathbf{u}_{POD}$. This yields the compressed and reconstructed primal based on the POD truncation and will be used in the continuation of the thesis.

3.2. Autoencoder - Neural Network Compression and Reconstruction

Primal data is also compressed and reconstructed using an autoencoder neural network. This autoencoder consists of two networks, an encoder and a decoder. The input of the encoder is nodal values. The encoder compresses the data and outputs the compressed data to the decoder. The decoder reconstructs the original primal data and outputs the reconstructed values. The input and output of the autoencoder are the same size.

The encoder neural network was constructed using a combination of both convolutional and linear layers. The convolutional layers are used in the initial layers of the encoder neural network. This is due to the size of the input being large. After some compression is done by the convolutional layers, linear layers are used to fully connect the last few layers such that the final compressed state of the primal is obtained. Fully connected layers are used here such that less information is lost during the final stages of the compression and to control the latent space size.

Similar to the encoder neural network, the decoder neural network consists of fully connected linear layers followed by convolutional layers. The amount of linear and convolutional layers is the same for the encoder and decoder.

Note that due to the fully connected linear layers, a different autoencoder is needed for each resolution. The architectures for the autoencoders can be found in Appendix A. The input of the encoder has a batch size of 32 time steps, a channel size that is equal to the number of variables, and the primal data in spatial points. The autoencoder compresses the primal input data to the size of latent space and reconstructs the primal data back to its original size.

The autoencoder neural network is trained using 10 batches of a batch size of 32. The batch size indicates that after 32 random time steps have been inputted, the autoencoder adapts its weights and biases. The number of batches indicates how many times the autoencoder will learn during one epoch. The training is continued for 1000 epochs and the model is saved every 50 epochs. The learning rate used for training is $1e-4$ or $5e-4$ depending on the PDE and uses the ADAM optimiser for training. The loss function used to train the autoencoder neural network is the MSE between the inputted data and the reconstructed data as can be seen in Equation 3.4. In Equation 3.4, n denotes the number of spatial samples.

$$\text{MSE} = \frac{\sum(\mathbf{u}_{\text{Reconstructed}} - \mathbf{u}_{\text{Inputted}})^2}{n} \tag{3.4}$$

To ensure that the training and testing data is statistically representative, the data is transformed to acquire more training data. The transformations that were applied are rotation and mirroring of data or a combination of both. By creating more training and testing data, the autoencoder can be trained and tested using more statistically representative data. This also helps against overfitting the autoencoder.

The hardware used to solve the PDEs, determine the POD and train the autoencoder, is a four-core, eight-thread Intel i7-7700HQ CPU with 8 GB of RAM and an Nvidia Quadro M1200 with 4 GB of GDDR5 VRAM. All training and testing of the neural network is done locally.

When the training and testing were done for the autoencoder, the saved models were called and used to obtain the compressed and reconstructed primal solution $\mathbf{u}_{\text{AE}}$.

3.3. Output-based Error Estimation

In this section, output-based error estimation will be elaborated upon. Output-based error estimation and in particular, adjoint-based error estimation is the primary way of determining local error estimates in this thesis. It can be used to compare the different compression and reconstruction methods. First, this section describes the derivation and implementation of the error estimation based on the adjoint solution. After this, the implementation of the previous sections is given in the error estimation framework.

Output-based error estimation uses the sensitivity of the output based on the numerical solution of the problem in order to obtain error estimates. The sensitivity of the output is investigated through the use of the adjoint solution of the CFD problem. The adjoint solution is obtained by solving the adjoint problem. The adjoint method and problem have been demonstrated and are explained in more detail in [5, 10, 40, 47].

This section focuses on the method to obtain the continuous adjoint problem for output-based error estimation based on the framework provided in [10].

The adjoint method considers a PDE of the form:

$$L\mathbf{u}=\mathbf{f} \quad \text{with} \quad \mathbf{R}(\mathbf{u})=L\mathbf{u}-\mathbf{f} \tag{3.5}$$

In Equation 3.5, L is the linear operator corresponding to an arbitrary PDE, **f** is a known forcing and u is the unknown solution. The residual is represented by $\mathbf{R}(\mathbf{u})$.

The adjoint operator L^* is defined as the relation between the primal operator and an adjoint set of solutions Ψ through the equation:

$$(L\mathbf{u},\Psi)=(L^*\Psi,\mathbf{u})$$

Here, the brackets represent the integrated inner product. To find the adjoint operator L^*, the left-hand side of the equation is filled in and rewritten such that the operators apply only on Ψ. The adjoint solution Ψ is used to find the sensitivity of a selected output $J(\mathbf{u})$ with regard to an incredibly small disturbance in the primal residual. This is seen in Equation 3.6. Based on this equation, it can be seen that the adjoint dictates how a change in the residual leads to a change in the output [10]. Equation 3.6 is the generalized form of the continuous adjoint equation.

$$J'(\delta\mathbf{u})=\int_\Omega \Psi^T\mathbf{R}'(\delta\mathbf{u})d\Omega \quad \forall \quad \delta\mathbf{u} \tag{3.6}$$

The adjoint operator L^* can also be derived by disturbing the PDE by an arbitrary fluctuation $\delta\mathbf{u}$ as seen in Equation 3.6. By completing the integration by parts for each term of Equation 3.6 and rewriting the terms such that the operators only work on the adjoint Ψ, the adjoint operator L^* is derived. The boundary conditions are then found by linking the different known boundary conditions from the primal problem to the adjoint boundary condition terms acquired through integration by parts. For non-linear problems, Equation 3.6 must be adapted such that the primal residual sensitivity is linearised about a given state $\mathbf{u}_0$. This leads to Equation 3.7.

$$J'[\mathbf{u}_0](\delta\mathbf{u})=\int_\Omega \Psi^T\mathbf{R}'[\mathbf{u}_0](\delta\mathbf{u})d\Omega \quad \forall \quad \delta\mathbf{u} \tag{3.7}$$

For unsteady problems, the time variation of the change in output and the change of the residual should be taken into account. Because of the discrete time stepping and finite element techniques used in this approach, the time integrals will be approximated using the discrete sum.

The derivation for error estimation is given in [10], and yields Equation 3.8, which is roughly approximated by Equation 3.9. In Equation 3.9, $\mathbf{u}_h^H$ denotes the coarse solution injected into the fine space using interpolation in space. The interpolation is done to achieve cheap fine residuals of the primal solution.

$$\delta J_{\text{est}} = J_h(\mathbf{u}_h) - J_H(\mathbf{u}_H) \tag{3.8}$$

$$\delta J \approx \delta J_{\text{est}} = -\int_{\Omega} \Psi_h^T \mathbf{R}_h(\mathbf{u}_h^H) d\Omega \tag{3.9}$$

Equation 3.9 is called the adjoint-weighted residual approach since the non-zero residual induced by the truncation error is weighted by the adjoint solution. Note that a linearization error $\mathcal{O}(\delta\mathbf{u}^2)$ is introduced in the estimate in Equation 3.7, due to the negligence of terms higher order $\delta\mathbf{u}$ terms. Additionally, because this linearization error might predominate when $\delta\mathbf{u}^2$ is large, non-linear problems may be far more vulnerable to under-resolved primal solutions. This means that the error estimates may be inexact [10].

The error can be localised by using Equation 3.9 to determine the adjoint weighted residual over a cell. Taking the absolute value of Equation 3.9, yields Equation 3.10 which gives the absolute local error estimate field on the fine space.

$$\varepsilon_{est} = \left|\Psi_h^T \mathbf{R}_h\left(\mathbf{u}_h^H\right)\right| \tag{3.10}$$

3.4. Proposed Method

Now that all compression and reconstruction techniques are discussed and the error estimation framework has been given, the final step is to combine the error estimation framework with the POD reconstructed solution $\mathbf{u}_{\text{POD}}$ and the autoencoder reconstruction $\mathbf{u}_{\text{AE}}$. This can be seen in Figure 3.1. In Figure 3.1, the error types introduced during the proposed method have been indicated by color. The

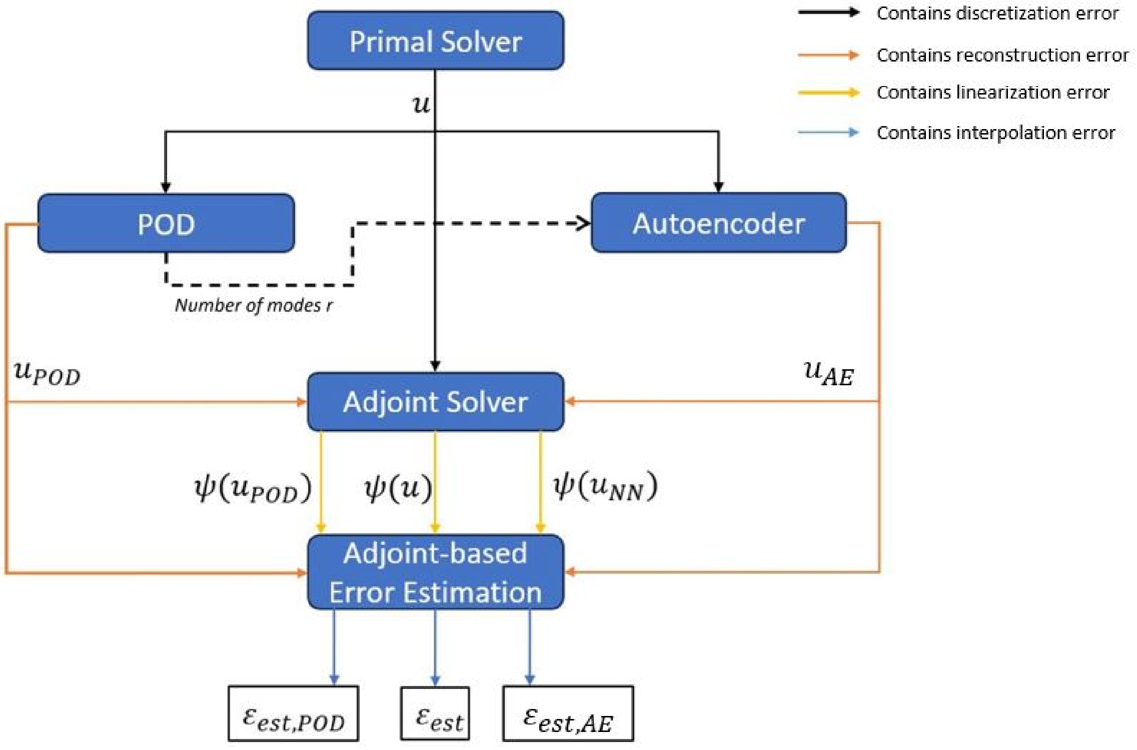

Figure 3.1: Schematic view of the proposed method with all errors introduced

compression and reconstruction techniques introduce a reconstruction error due to lossy compression. The adjoint solver uses the linearized versions of PDEs which results in a linearization error. Finally, the adjoint-based error estimation method uses the coarse solution injected into the fine space using interpolation in space $\mathbf{u}_h^H$. This introduces an interpolation error.

Based on this approach, the influence of both compression and reconstruction techniques on the adjoint-based error estimation can be compared with the original, non-reconstructed, adjoint-based error estimates. This is done for two sets of PDEs. The first PDE is the 1D unsteady viscous Burgers equation in Chapter 4. The second set of PDEs is the Navier-Stokes equations for the lid-driven cavity flow in Chapter 5.

4 Validation: 1D Unsteady Viscous Burgers

In this chapter, the 1D unsteady viscous Burgers equation will be used as validation for the primal solver and error estimation framework using the method of manufactured solutions found in [17]. The validation for the adjoint solver is done by comparing the adjoint to solutions presented in [1]. First, the governing equations are presented in Section 4.1 as well as the solutions obtained from the finite element method and the manufactured solution. After this, the POD method is validated by applying it to the discrete primal solution in Section 4.2. Next, the autoencoder will be applied to the discrete primal solution in Section 4.3. After this, the adjoint equations will be presented and solved in Section 4.4. Lastly, the error estimates are shown in Section 4.5. The results presented in this chapter are for the spatial resolution of 32 points. The solutions of the 1D Burgers equation for different spatial resolutions can be found in Appendix B. The solutions have been generated for the spatial resolutions $8, 16, 32, 48, 64, 96$, and 128 points.

4.1. Primal Equations and Solution for 1D Burgers equation

Consider the one-dimensional Burgers equation over a space-time domain $\Omega : [0,1] \times I : [0,20]$. Many applications that require shock wave propagation in viscous flows or idealized turbulence employ the Burgers equation as a mathematical model [48]. The Burgers equation can be seen in Equation 4.1.

$$\begin{aligned} \text{PDE:} \quad & \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\frac{\partial \mathbf{u}}{\partial x} - \nu \frac{\partial^2 \mathbf{u}}{\partial x^2} = f, \\ \text{BC:} \quad & \mathbf{u}(x,t) = 0 \quad \text{on } \partial\Omega, \\ \text{IC:} \quad & \mathbf{u}(x,0) = 0 \quad \text{on } \Omega \end{aligned} \tag{4.1}$$

In Equation 4.1, **u** is the solution with homogeneous Boundary Conditions (BCs) $\mathbf{u}(0,t) = \mathbf{u}(1,t) = 0$ on the boundary $\partial\Omega$. The initial condition (IC) is $\mathbf{u}(x,0) = 0$. $\nu = \frac{1}{Re} = \frac{1}{100}$ is the viscosity coefficient and f $\in \mathbb{R}$ is a known forcing term or source function. Note that the Burgers problem in this thesis is used to investigate the proposed method in one dimension. However, the considered approaches can also be applied to multi-dimensional problems. A manufactured solution corresponding to f is $\mathbf{u}_{\text{MMS}}(x,t) = \sin^2(\pi t)\sin(\pi x)$.

The QoI for this case is $\bar{J} = \frac{1}{T}(\sin(\pi x), \mathbf{u})_{\Omega \times I}$, where the brackets denote the integrated inner product. To get the exact value of the QoI, the QoI is calculated using the manufactured solution which results in $\bar{J} = 0.25$.

The solution for the 1D Burgers equation can be seen in Figure 4.1. Figure 4.1 shows the solution for a spatial resolution of 32 points. The manufactured solution with the same spatial resolution can be seen in Figure 4.2. For all the numerical results a time step Δt of 0.001 is used to prevent temporal discretization errors influencing the solution [17].

By looking at the discrete primal solutions from low to high spatial resolution in Section B.1.1, the absolute error between the manufactured and discrete primal solution decreases. This is seen by plotting the absolute error between the discrete primal and the manufacture solution as is shown in Figure 4.3 and Section B.2.1. Figure 4.3 shows that the absolute error has periodic behaviour and is centred around the middle of the domain for the later time steps. For the lower spatial resolutions of the discrete primal shown in Section B.2.1, the top left corner of the absolute error plots shows a discretisation error which fades as the spatial resolution goes up. This could be due to some high-wave number flow components that could not be captured by coarse spatial resolutions. This resulted in the high absolute error located in the top left corner for lower spatial resolutions.

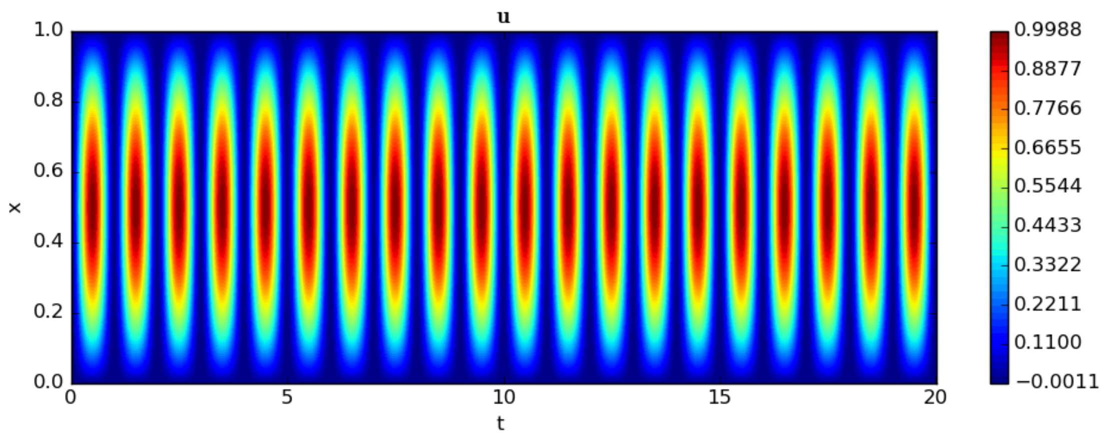

Figure 4.1: The solution of the 1D burgers equation of $N_H = 32$

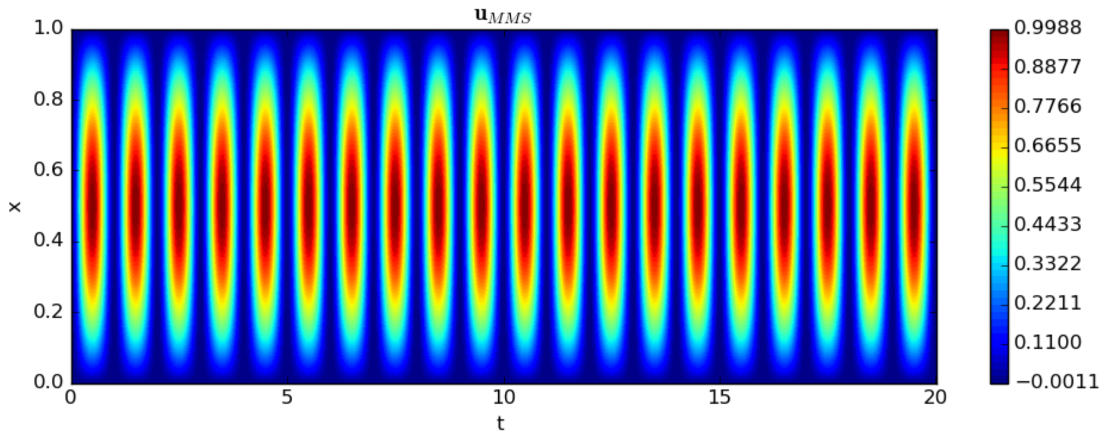

Figure 4.2: The reference solution of the 1D burgers equation of $N_H = 32$

From Figure 4.1, Figure 4.2, Figure 4.3, and the figures in Appendix B, it can be seen that a more accurate solution is achieved when increasing the spatial resolution. This is also indicated by the Mean Squared Error (MSE) which is defined by Equation 4.2 and can be seen in Figure 4.4. The MSE error decreases and stabilises as the spatial resolution increases.

$$MSE(\mathbf{u},\mathbf{u}_{\text{MMS}}) = \frac{1}{N}\sum(\mathbf{u}-\mathbf{u}_{\text{MMS}})^2 \tag{4.2}$$

4.2. Compression and reconstruction using POD for 1D Burgers equation

Now that the primal solution is known, the primal solution is compressed using the POD approach, explained in Section 3.1. The POD compression respects the total mode energy level of retaining 99% and is applied to the fluctuations of the discrete primal solution. When looking at the chosen manufactured

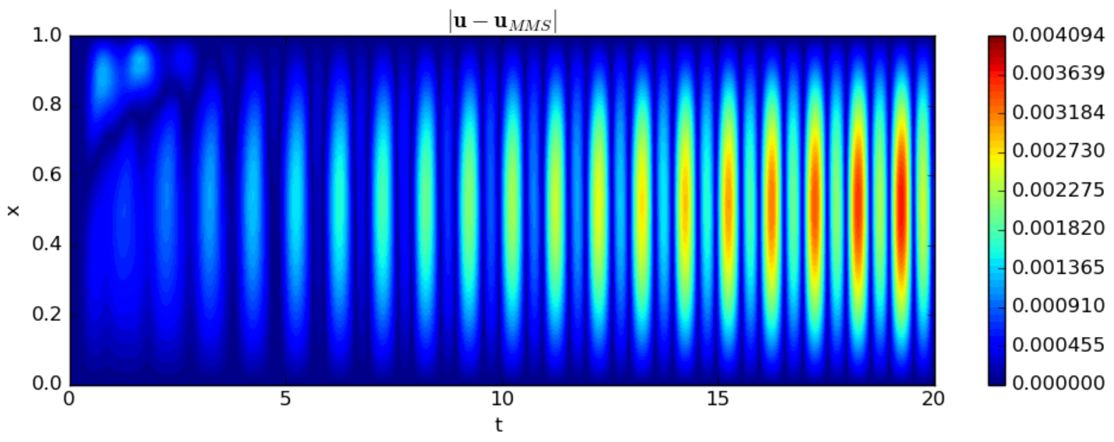

Figure 4.3: The solution error of the 1D burgers equation of $N_H = 32$

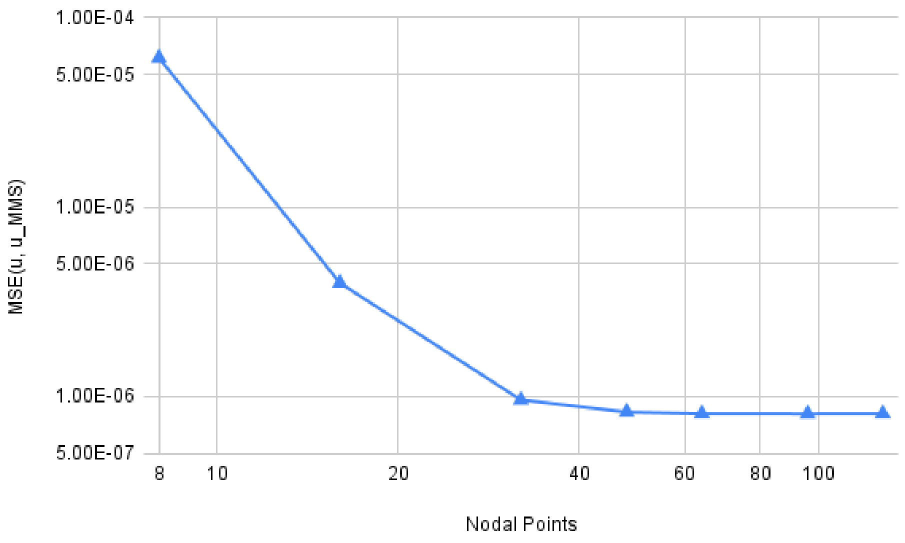

Figure 4.4: 1D Burgers MSE of the discrete primal for different spatial resolutions

solution, it can be seen that this solution can be reconstructed with just two modes. The mode energy per POD mode can be seen in Figure 4.5. It can be seen that most of the energy is retained in the first POD mode for all spatial resolutions.

For the different spatial resolutions, the number of modes r and the achieved compression ratio CR can be seen in Table 4.1. The MSE is defined by Equation 4.3 and can be seen in Figure 4.6. In Figure 4.6, the MSE decreases as the spatial resolution goes up. Due to increasing spatial resolution the POD uses more values to reconstruct the discrete primal. This behavior is expected since the POD is able to reconstruct the discrete primal solution exactly using only two modes as can be seen from Table 4.1. The rate of convergence can be seen to be between 4 and 5 for the MSE of the POD.

$$MSE(\mathbf{u}_{\text{POD}},\mathbf{u}) = \frac{1}{n}\sum(\mathbf{u}_{\text{POD}} - \mathbf{u})^2 \tag{4.3}$$

The compression ratio for POD for the one-dimensional Burgers equation CR_{POD} is defined by Equation 4.4 where n_{var} is the number of variables, which is equal to 1 in this case.

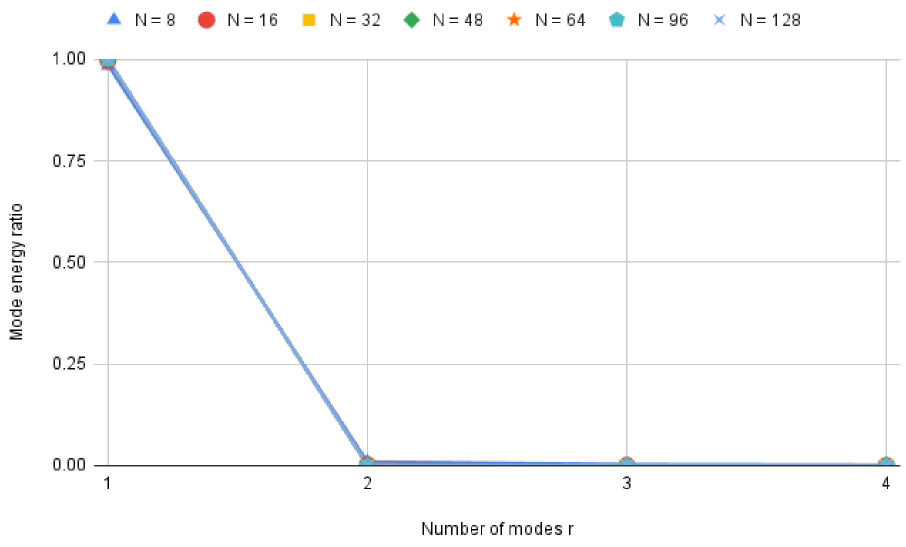

Figure 4.5: Mode energy for each POD mode

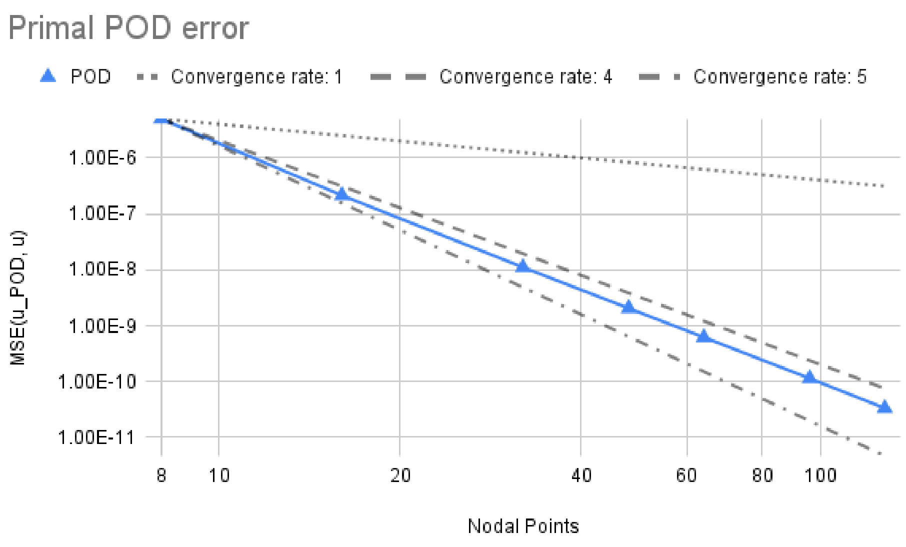

Figure 4.6: 1D Burgers MSE of the POD primal for different spatial resolutions on the coarse space

$$CR_{\text{POD}} = \frac{\text{Input data size}}{\text{POD output data size}} = \frac{\text{Time resolution} \times \text{Spatial resolution of input}}{\text{Size}(\mathbf{U}_r) + \text{Size}(\boldsymbol{\Sigma}_r) + \text{Size}(\mathbf{V}_r^T) + n_{var}} \tag{4.4}$$

The compressed and reconstructed fields based on the POD can be seen in Figure 4.7. The reconstructed fields for the other spatial resolutions can be found in Section B.1.2. The reconstructed solution looks qualitatively the same as the discrete primal solution.

The absolute error between the POD reconstructed solution and the discrete primal solution can be seen in Figure 4.8. Again in the top left corner of the figure, the same discretisation error is seen as was seen previously in the discrete primal solution. The absolute error fields for the other spatial resolutions can be seen in Section B.2.2. When comparing the absolute error fields from low to high spatial resolution, it can be seen that for a spatial resolution of 96 and upwards, the reconstructed solution is equal to the discrete primal solution. This was achieved using two POD modes as can be seen in Table 4.1, which indicates that the POD approach is validated.

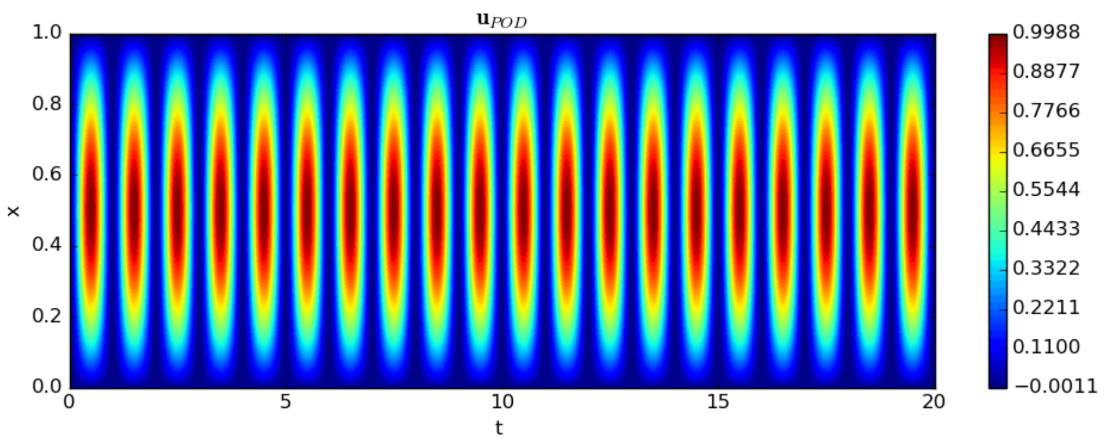

Figure 4.7: POD reconstructed 1D Burgers solution with a spatial resolution of $N_H = 32$

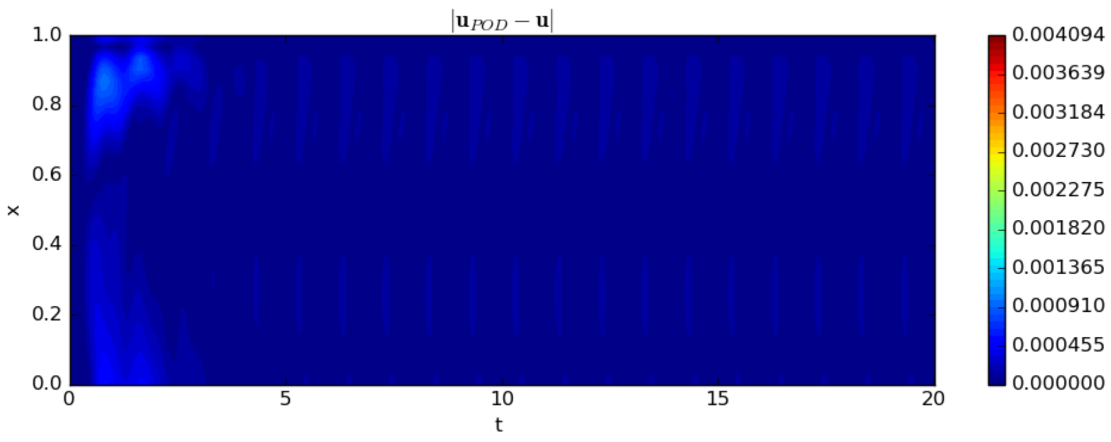

Figure 4.8: Difference between exact and POD reconstructed 1D Burgers solution with a spatial resolution of $N_H = 32$

Spatial resolution	Optimal rank r	Compression Ratio CR_{POD}
8	2	3.999100247
16	2	7.996601614
32	2	15.98681154
48	2	23.97063744
64	2	31.94808695
96	2	47.88388738
128	2	63.79427375

Table 4.1: POD Compression information for different spatial resolutions for the 1D Burgers equation

4.3. Compression and reconstruction using NN for 1D Burgers equation

Now, the primal is compressed using an autoencoder as well. The autoencoder is constructed using the method given in Section 3.2. As was noted before, due to the fully connected linear layers, the autoencoder has a different architecture which can be found in Appendix A. The compression ratio of the autoencoder can be calculated using Equation 4.5.

$$\begin{aligned} CR_{\text{AE}} &= \frac{\text{Input data size}}{\text{Encoder output data size} + \text{Model size}} \\ &= \frac{\text{Time resolution} \times \text{Spatial resolution of input}}{\text{Time resolution} \times \text{Spatial resolution of encoder output} + \text{Number of Model parameters}} \end{aligned} \tag{4.5}$$

The autoencoder is designed to have a latent space size of one. This means that the autoencoder compresses the discrete primal solution at a single time step to just one value and reconstructs it back to its original size to obtain the autoencoder reconstructed solution $\mathbf{u}_{\text{AE}}$.

The train and test losses for the autoencoder for the spatial resolution of 32 can be seen in Figure 4.9. In Figure 4.9, it can be seen that an accuracy of 10^{-6} is reached with this autoencoder. For the last 300 epochs, oscillations can be seen in the test and train losses. This is due to the learning rate being too high at this stage of training. A high learning rate will result in the local minimum being overshot which results in the oscillations seen in Figure 4.9. Despite the oscillations, the autoencoder model is still applicable to the problem. It should be noted that more model parameters will increase the computational cost of training and running the autoencoder. Even though the computational cost is beyond the scope of this thesis, it should not be overlooked.

$$MSE(\mathbf{u}_{\text{AE}}, \mathbf{u}) = \frac{1}{n} \sum (\mathbf{u}_{\text{AE}} - \mathbf{u})^2 \tag{4.6}$$

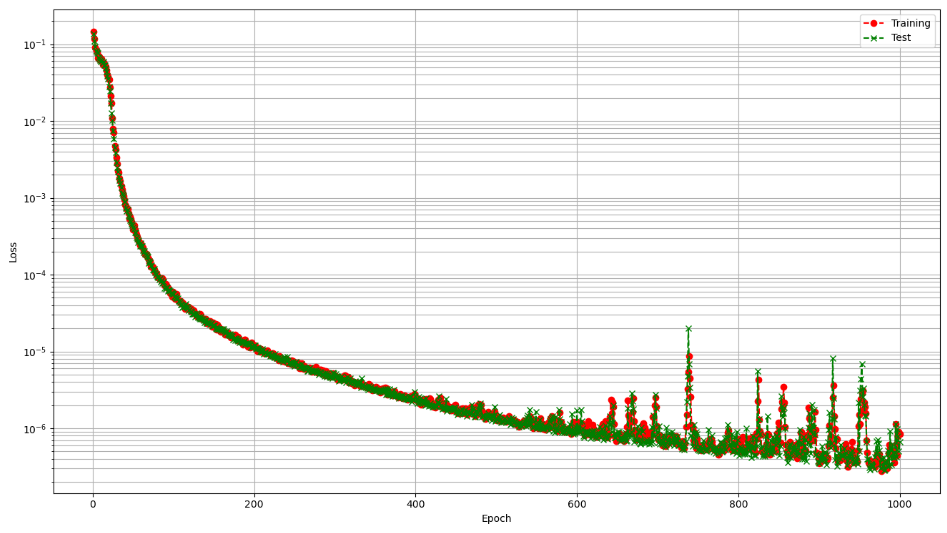

Figure 4.9: Train and test losses of the autoencoder for a spatial resolution of $N_H = 32$ for the 1D Burgers equation

The velocity field for a spatial resolution of 32 can be seen in Figure 4.11 and looks qualitatively the same as the discrete primal solution. More velocity fields of each autoencoder can be found in Section B.1.3. When looking at the absolute error of the autoencoder reconstructed solution and the discrete primal in Figure 4.12, it can be seen that the errors introduced return periodically. These errors stem from the latent space being too small. The latent space, which is the encoder output, consists of too few variables for the decoder to reconstruct the solution accurately. This can be solved by increasing the latent space however, this will also increase the compression ratio.

For the different spatial resolutions the achieved compression ratio CR_{AE} and total modal parameters can be seen in Table 4.2. It should be noted that the compression ratios for a spatial resolution of 48, 96 and 128 are significantly lower. When training for these particular spatial resolutions, the loss function would not decrease to an acceptably low value. Therefore, the latent space of these spatial resolutions has been increased with one variable. This resulted in the lower compression ratios.

The MSE is defined by Equation 4.6 and is visualized in Figure 4.10. It can be seen that the MSE first decreases and then stabilizes whereafter the MSE actually increases for the higher spatial resolutions. For higher spatial resolutions, the autoencoder needs to reconstruct more values from the same latent space. This results in the errors increasing for higher spatial resolutions which are also seen in the trend of the MSE error. Naturally, the MSE will decrease for higher spatial resolutions due to the solution of the discrete primal being more accurate. Therefore, the MSE first decreases as the refinement of the solution is still dominant. However, for a spatial resolution of 16 to 64, the MSE is balanced out by the error introduced by the reconstruction of the autoencoder, and the error decreases due to refinement. This is seen by the plateau of this region. After that, for spatial resolutions of 96 and above, the MSE error is dominated by the reconstruction error introduced by the autoencoder.

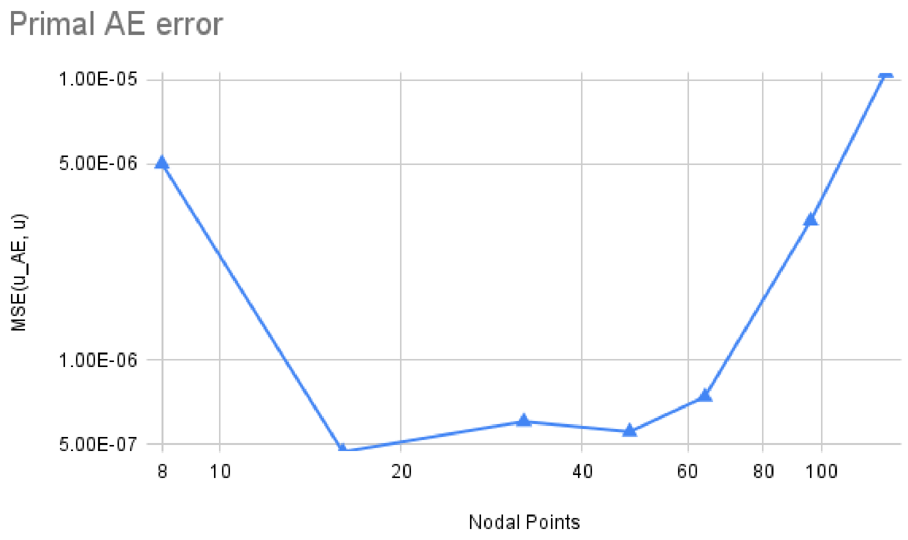

Figure 4.10: 1D Burgers MSE of the autoencoder primal for different spatial resolutions on the coarse space

Spatial resolution	Model Parameters	Compression Ratio CR_{AE}
8	20018	3.998300807
16	20090	7.982240403
32	20318	15.87420323
48	21078	15.71787819
64	19790	32.16968661
96	29507	27.62370341
128	26941	38.2434011

Table 4.2: Autoencoder compression information for different spatial resolutions for the 1D Burgers equation

4.4. Adjoint Equations and Solution for 1D Burgers equation

Now that **u**, and $\mathbf{u}_{\text{AE}}$ are known, the adjoint can be calculated based on these solutions. To do so, the adjoint equation first needs to be derived. From Equation 4.1, the adjoint equation can be derived. This is done using the framework provided by [10]. Since the one-dimensional burgers equation is a non-linear equation, the adjoint equation will be derived around a linearized state $\mathbf{u}_0$. From Equation 3.7, it can be seen that the derivative of the residual with respect to $\delta\mathbf{u}$, which is an infinitesimally small perturbance applied to **u**, is needed. This is seen in Equation 4.7.

$$\mathbf{r}'[\mathbf{u}_0](\delta\mathbf{u}) = \frac{\partial(\delta\mathbf{u})}{\partial t} + \mathbf{u}_0\frac{\partial(\delta\mathbf{u})}{\partial x} + \delta\mathbf{u}\frac{\partial\mathbf{u}_0}{\partial x} - \nu\frac{\partial^2(\delta\mathbf{u})}{\partial x^2} \tag{4.7}$$

Now to get the sensitivity of output J, Equation 4.7 is multiplied with the adjoint Ψ as can be seen in Equation 4.8 and can be partially integrated.

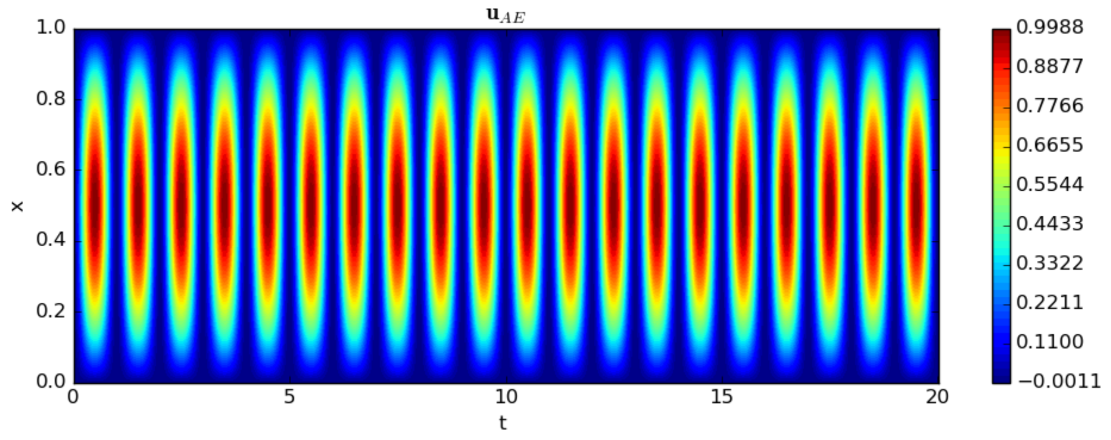

Figure 4.11: Autoencoder reconstructed 1D Burgers solution for a spatial resolution of $N_H = 32$

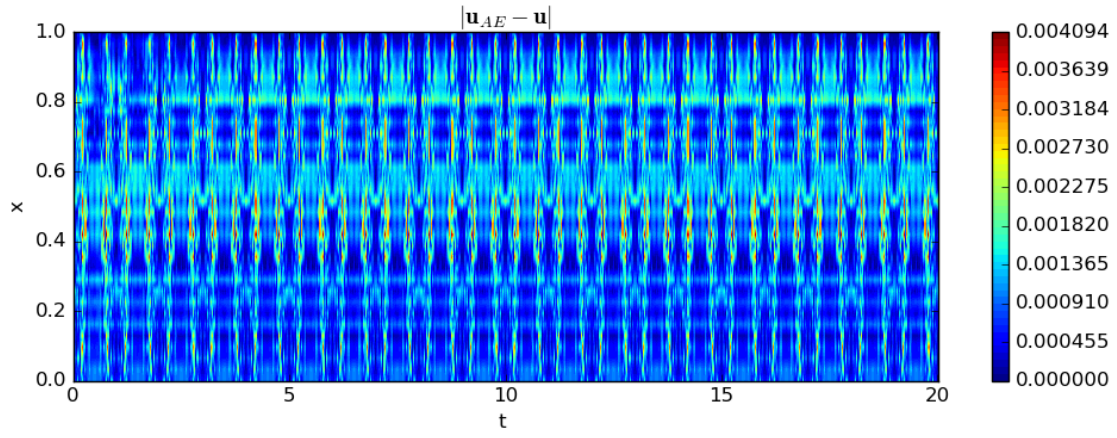

Figure 4.12: Difference between exact and Autoencoder reconstructed 1D Burgers solution with a spatial resolution of $N_H = 32$

$$\begin{aligned} J'[\mathbf{u}_0](\delta\mathbf{u}) &= \int_{\Omega\times I} \Psi^T \mathbf{r}'(\delta\mathbf{u}) d\Omega \quad \forall \quad \delta\mathbf{u} \\ &= \int_I \Psi \frac{\partial(\delta\mathbf{u})}{\partial t} dT + \int_\Omega \Psi(\mathbf{u}_0 \frac{\partial(\delta\mathbf{u})}{\partial x} + \delta\mathbf{u}\frac{\partial \mathbf{u}_0}{\partial x} - \nu \frac{\partial^2(\delta\mathbf{u})}{\partial x^2}) d\Omega \\ &= [u\Psi]_I - \int_I u \frac{\partial\Psi}{\partial t} dT + ([\mathbf{u}_0 \delta\mathbf{u}\Psi]_\Omega - \int_\Omega \mathbf{u}_0 \delta\mathbf{u} \frac{\partial\Psi}{\partial x} d\Omega) + \int_\Omega \delta\mathbf{u} \frac{\partial\mathbf{u}}{\partial x} \Psi d\Omega \\ &- ([\nu \frac{\partial(\delta\mathbf{u})}{\partial x} \Psi]_\Omega - ([\nu\delta\mathbf{u}\frac{\partial\Psi}{\partial x}]_\Omega - \int_\Omega \nu\delta\mathbf{u}\frac{\partial^2\Psi}{\partial x^2} d\Omega)) \end{aligned} \tag{4.8}$$

The BCs and IC for **u** can be applied for δ**u** since fixed values will not vary when a perturbance is introduced. If δ**u** is then replaced by **u**, Equation 4.8 can be simplified to Equation 4.9.

$$J'[\mathbf{u}_0](\mathbf{u}) = -\int_I \mathbf{u}\frac{\partial\Psi}{\partial t} dT - \int_\Omega \mathbf{u}_0 \mathbf{u} \frac{\partial\Psi}{\partial x} d\Omega + \int_\Omega \mathbf{u}\Psi \frac{\partial\mathbf{u}_0}{\partial x} d\Omega - \int_\Omega \nu\mathbf{u}\frac{\partial^2\Psi}{\partial x^2} d\Omega + [\mathbf{u}\Psi]_I + [\nu\frac{\partial\mathbf{u}}{\partial x}\Psi]_\Omega \tag{4.9}$$

Based on the last two terms of Equation 4.9, the BCs and IC can be derived for the adjoint. By omitting the integrals and **u**, the adjoint equation as seen in Equation 4.10, is derived.

$$\begin{aligned} \text{Adjoint:} &\quad \frac{dJ}{d\mathbf{u}} = -\frac{\partial \Psi}{\partial t} + \Psi \frac{\partial \mathbf{u}_0}{\partial x} - \mathbf{u}_0 \frac{\partial \Psi}{\partial x} - \nu \frac{\partial^2 \Psi}{\partial x^2} = \sin(\pi x) = g(x) \\ \text{BC:} &\quad \Psi(x,t) = \Psi(x,t) = 0 \quad \text{on } \partial\Omega, \\ \text{IC:} &\quad \Psi(x,20) = 0 \quad \text{on } \Omega \end{aligned} \tag{4.10}$$

Note that the minus sign before the partial time derivative in Equation 4.10 indicates that the time stepping will go backward and that the initial condition is set at time $T = 20$. The right-hand side of the equation is obtained by omitting the time averaging, the integral, and **u** from the definition of $J = \frac{1}{T}\int_\Omega \sin(\pi x)\mathbf{u} d\Omega$.

To calculate the adjoint solution based on $\mathbf{u}_{\text{AE}}$, **u** is simply replaced in Equation 4.10 as the adjoint equation does not change.

All adjoint solutions are solved on a twice as refined spatial resolution as the primal solution. The solution can be seen in Figure 4.13. The solution of the adjoint is only calculated up to $T/2 = 10$. The adjoint solutions for the fine spatial resolutions of $16, 32, 64, 96$ and 128 can be seen in Section B.3. The adjoint solutions are calculated for the discrete primal solution and the compressed and reconstructed primal solution using the autoencoder. The results for the adjoint for all spatial resolutions can be seen in the subsections of Section B.3. The adjoint solutions for a spatial resolution of 32 can be seen in Figure 4.13 and Figure 4.14 for the discrete primal and the compressed and reconstructed primal solution using the autoencoder respectively. The adjoint solutions for all primal solutions look qualitatively similar and behave similarly. This is consistent with the results presented in [1].

The absolute error between the adjoint solution and the autoencoder reconstructed primal adjoint is seen in Figure 4.15. It can be seen that the error grows as the solution moves back in time. This is a result of the error being added by the autoencoder. Due to the increasing maximum value of the adjoint, the absolute error of the adjoint increases as well. Additionally, on a local level, there are also errors present that behave like noise. This is seen by the absolute error regions fluctuating when going backwards in time.

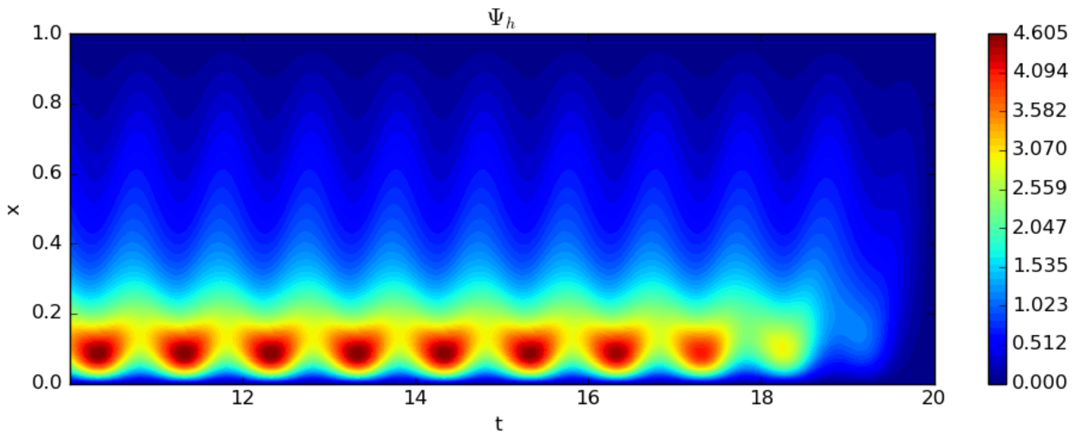

Figure 4.13: Adjoint for $N_h = 32$ for the 1D Burgers equation

The MSE between the autoencoder adjoint and the discrete primal adjoint is calculated using the Equation 4.11. The MSE is displayed in Figure 4.16. It can be seen that the error increases when spatial resolution increases. This is because of the reconstruction error introduced by the autoencoder. Since the autoencoder primal solution is inserted into the adjoint equation as the linearised state and the gradient of the linearized state, the error introduced by the autoencoder will contribute twice to the calculation of the adjoint solution. While the discrete primal gets more accurate for higher spatial resolutions and is better linearised because of this, the autoencoder gets less accurate for higher spatial resolutions, and linearisation is worsened.

$$MSE(\Psi_{\text{AE}}, \Psi) = \frac{1}{n}\sum(\Psi_{\text{AE}} - \Psi)^2 \tag{4.11}$$

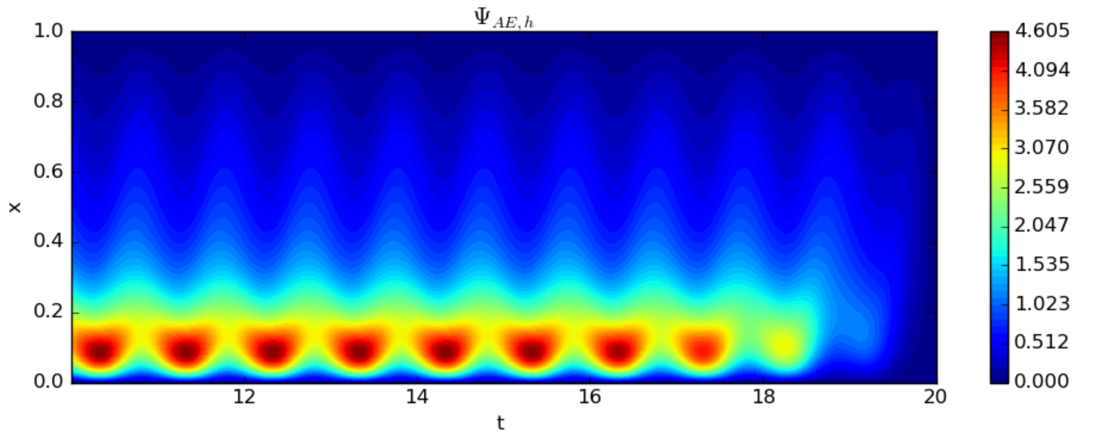

Figure 4.14: Autoencoder adjoint for $N_h = 32$ for the 1D Burgers equation

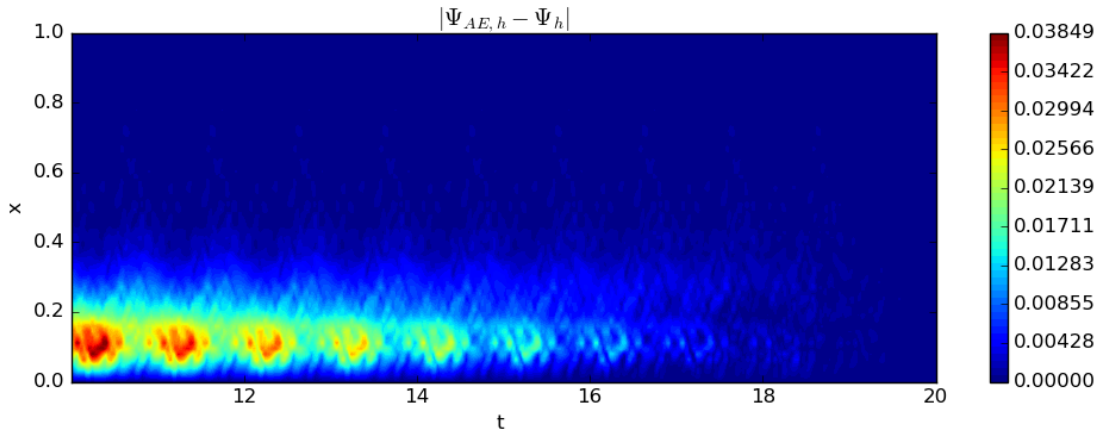

Figure 4.15:
Difference between adjoint based on the discrete primal and the autoencoder adjoint for $N_h = 32$ for the 1D Burgers equation

4.5. Adjoint-based Error Estimates for 1D Burgers equation

To find the error estimate δJ_{est} from Equation 3.9, the adjoint weighted residual needs to be integrated over the spatial and temporal domain $\Omega \times I$. This can be seen in Equation 4.12. Note that since the chosen QoI is an integral in both time and space, the error estimation also needs to be done in time and space. This is done by integrating over time in Equation 4.12.

$$\delta J_{\text{est}} = -\int_I \int_\Omega \Psi_h^T \mathbf{R}_h(\mathbf{u}_h^H) d\Omega dT \tag{4.12}$$

In Equation 4.12 the adjoint, residual and the primal solution are defined on the fine space h. The fine space h is taken to be twice as refined as the coarse space H. The adjoint solutions obtained from Section 4.4 have been calculated for the fine space h using $\mathbf{u}_h$, however, all primal solutions obtained thus far, are still defined on the coarse space H. The solution for the primal $\mathbf{u}_H$ as well as the compressed and reconstructed primal obtained from the autoencoder $\mathbf{u}_{\text{AE},H}$, will be interpolated to obtain the coarse solution injected into the fine space. This yields $\mathbf{u}_h^H$ and $\mathbf{u}_{\text{AE},h}^H$ which will be used in the error estimation method.

The residual of the primal on the fine space is given by Equation 4.13 which is derived from Equation 4.1. Note that the forcing f also needs to be defined on the fine space h.

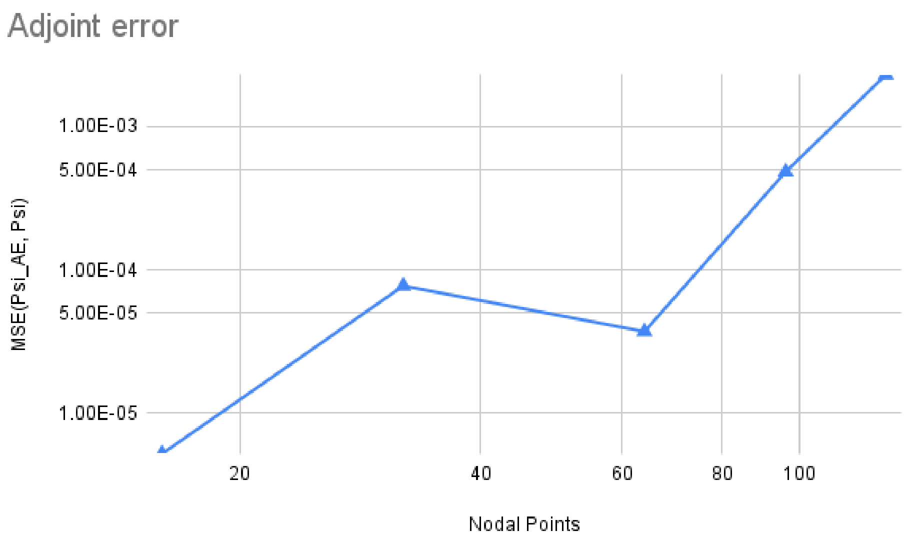

Figure 4.16: 1D Burgers MSE of the autoencoder adjoint for different spatial resolutions on the fine space

$$\mathbf{R}_h(\mathbf{u}_h^H) = \frac{\partial \mathbf{u}_h^H}{\partial t} + \mathbf{u}_h^H \frac{\partial \mathbf{u}_h^H}{\partial x} - \nu \frac{\partial^2 \mathbf{u}_h^H}{\partial x^2} - f_h \tag{4.13}$$

The solution for the residual of the primal using the coarse solution and coarse autoencoder solution injected into the fine space can be seen in Figure 4.17 and Figure 4.18, respectively. The solution for the other spatial resolutions can be found in Appendix B. In Figure 4.17, it can be seen that there are some small differences in the field. These are due to the introduced interpolation errors added during the injection of the coarse solution into the fine space. In Figure 4.18, besides the interpolation error, the reconstruction error from the autoencoder can be seen as well. The reconstruction errors make up the highest error values which return periodically as was the case for the autoencoder primal.

Finally, the absolute error between the residual and the autoencoder residual can be seen in Figure 4.19 and in Appendix B for the other spatial resolutions. This shows the periodic behaviour as well which is expected due to the errors made during autoencoder reconstruction of the primal. The MSE has been plotted as well in Figure 4.20. It can be seen that the error decreases with a rate of convergence between 1 and 2. Since the residual is calculated using the coarse solution injected into the fine space, and the reconstructed coarse solutions of the autoencoder have relatively fewer reconstruction errors, the value of the MSE residual error is seen to decrease when spatial resolution is increased.

The obtained residual on the fine space, together with the fine adjoint can then be filled in into Equation 4.12 to obtain the error estimates.

Since $\bar{J} = 0.25$, the value of $\bar{J}_H$ can be computed and compared as well. This can be done using the definition of the QoI. Based on Equation 3.8, the true error estimate can be computed. This error estimate can be compared with the adjoint-based error estimate from Equation 4.12.

The localised error estimate on the fine space for each element is retrieved from Equation 4.14 and can be seen in Figure 4.21 and Figure 4.22 using the discrete primal and the autoencoder primal, respectively a log scale is used to better see the differences. All other spatial resolutions can be found in Appendix B. It can be seen that the local error estimates using the autoencoder have higher local error estimates compared to the error estimates based on the discrete primal. The same periodic pattern that the residual displayed can be seen for the local error estimates as well. This similarity in behaviour is due to the autoencoder reconstruction error. The absolute error between the local error estimates is seen in Figure 4.23 and in Appendix B for all other spatial resolutions. In Figure 4.23, similar behaviour as in Figure 4.22 is seen due to the difference in local error estimate values being orders of magnitudes larger for the autoencoder primal. This results in the absolute error plot being dominated by the autoencoder residual solution. Additionally, the MSE of the local error estimates has been plotted in Figure 4.24. This is seen to resemble the behaviour seen in Figure 4.20 rather than the behaviour in Figure 4.16. This is

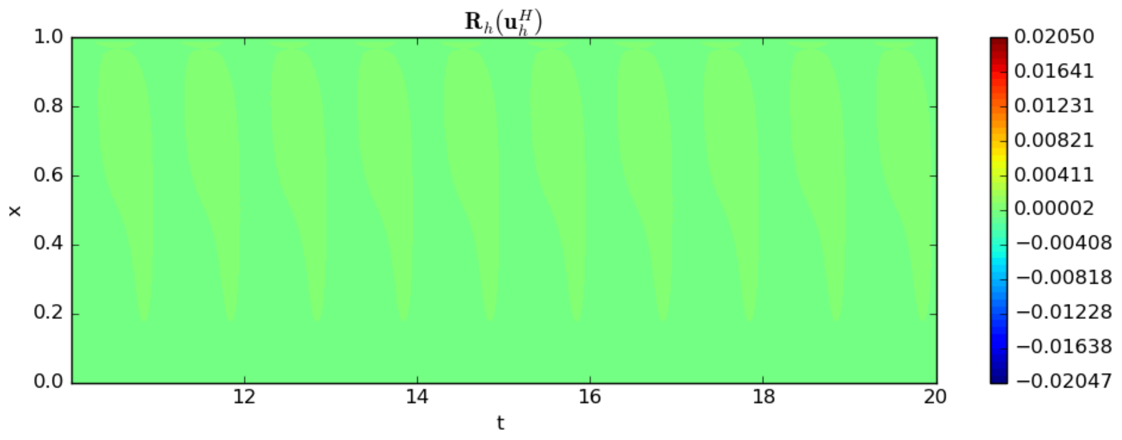

Figure 4.17: 1D Burgers residual using the coarse solution injected into the fine space for $N_h = 32$

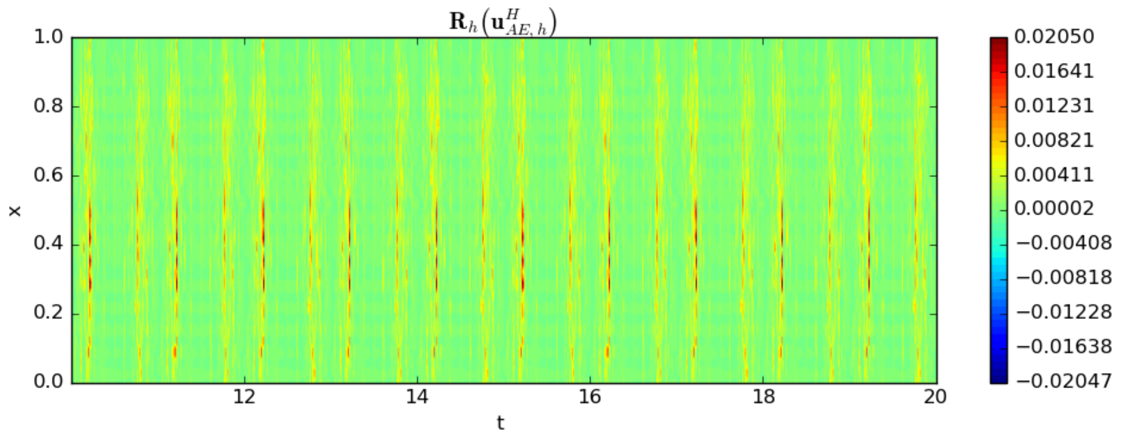

Figure 4.18: 1D Burgers autoencoder residual using the coarse solution injected into the fine space for $N_h = 32$

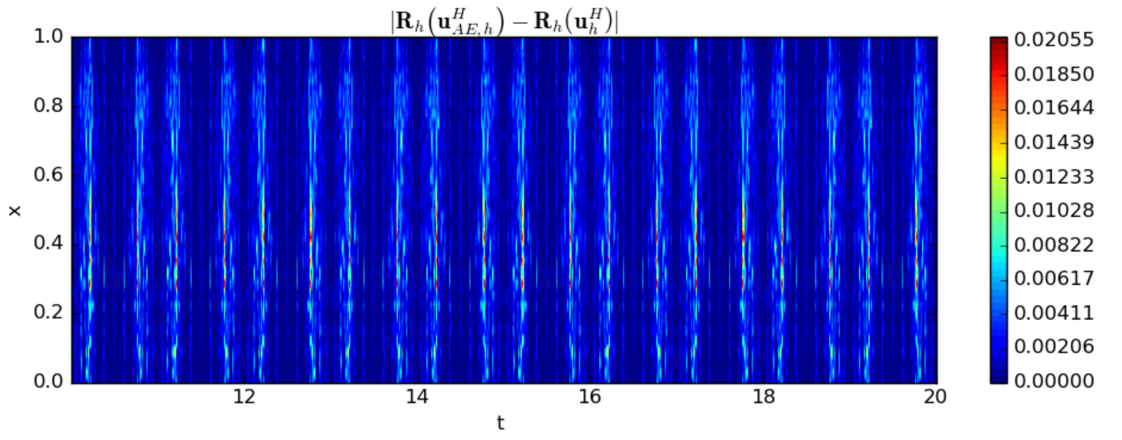

Figure 4.19:
Difference between residual based on the discrete primal and the autoencoder adjoint for $N_h = 32$ for the 1D Burgers equation

explained by the residual being relatively low where the adjoint solution is high. As the highest adjoint values are located near $x = 0.1$ and the highest residual values centered around the middle of the spatial

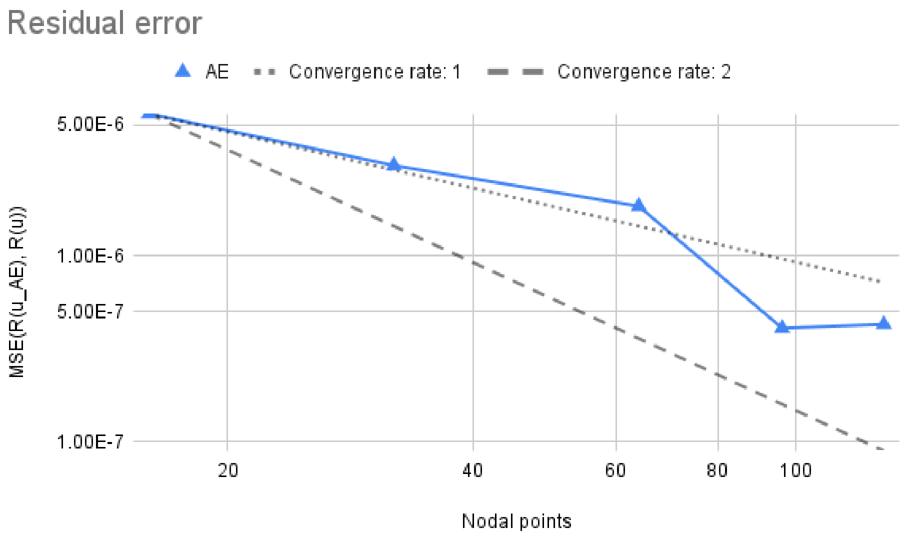

Figure 4.20: 1D Burgers MSE of the residual for different spatial resolutions on the fine space

domain, the contribution of the adjoint seems to be dampened due to the residual values being low in the regions where the adjoint is high. This results in the MSE of the local error estimates behaving similarly to the residual MSE. It can be seen, however, that the rate of convergence is closer to 1, which is lower than the convergence rate of the residual.

$$\varepsilon_{est} = \left|\Psi_h^T \mathbf{R}_h\left(\mathbf{u}_h^H\right)\right| \tag{4.14}$$

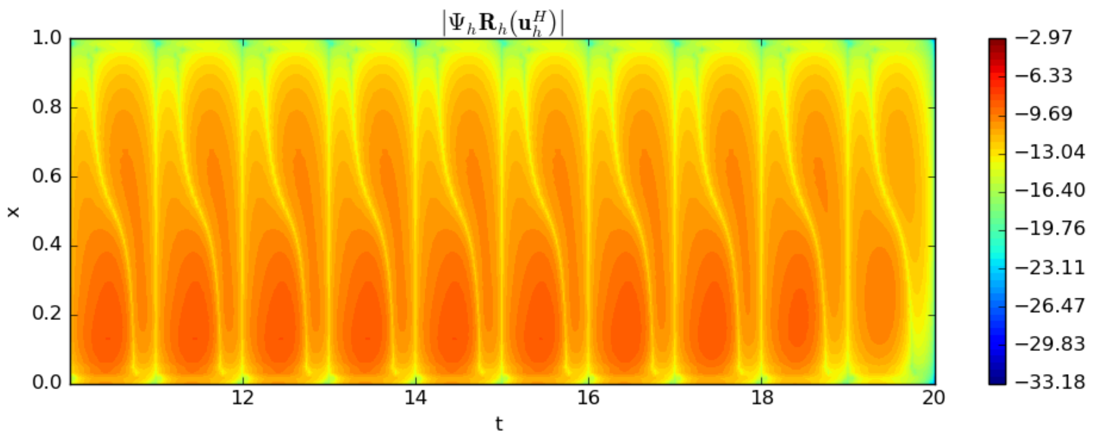

Figure 4.21: Local error estimates for $N_H = 32$ for the 1D Burgers equation

The QoI based on the coarse primal $\bar{J}_H$, the estimated values of $\bar{J}$, the adjoint-based error estimate $\delta\bar{J}_{\text{est}}$ and the deficiency in the error estimate $d\delta\bar{J}_{\text{est}}$ compared to the true error can be seen in Figure 4.25, Figure 4.26, Figure 4.27 and Figure 4.28 respectively. The definition of the deficiency in the error estimate is defined in Equation 4.15.

$$d\delta\bar{J} = \delta\bar{J}_{\text{true}} - \delta\bar{J}_{\text{est}} \quad \text{with} \quad \delta\bar{J}_{\text{true}} = \bar{J} - \bar{J}_H \tag{4.15}$$

From Figure 4.25, it can be seen that as the spatial resolution increases, the QoI approaches the exact value of 0.25. In Figure 4.26, it can be seen that the estimated QoI tends towards the exact value of 0.25 as the spatial resolution gets refined. It is first overestimated however, corrects itself as the spatial

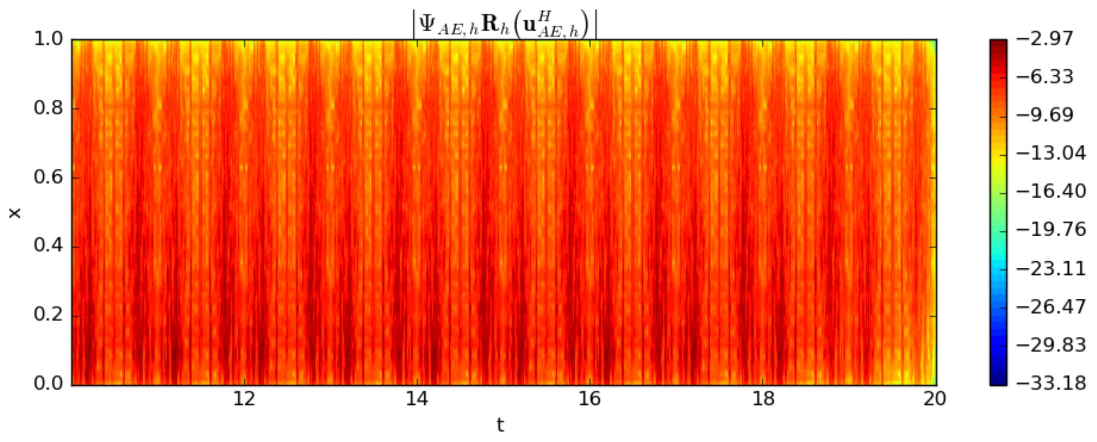

Figure 4.22: Local error estimates based on autoencoder reconstructed primal for $N_H = 32$ for the 1D Burgers equation

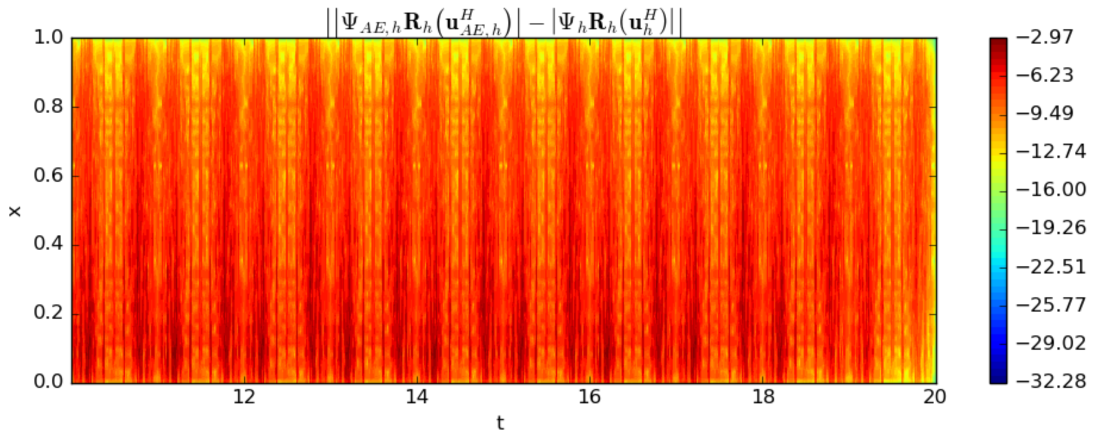

Figure 4.23: Difference
in local error estimates between discrete primal and autoencoder reconstructed primal for $N_H = 32$ for the 1D Burgers equation

resolution increases. It can be seen from Figure 4.27 that the autoencoder error estimate is accurate for the lower spatial resolutions and deviates more for the higher spatial resolutions due to reconstruction being more accurate for lower spatial resolutions. This is also seen in Figure 4.29 where the discrete primal and autoencoder error estimates are normalised with the true error estimates. From Figure 4.29, the similarity between the true error and the error estimates can be seen. It should be noted that in Figure 4.29, the relative distance from a value of 1 measures how similar the values are. Therefore, when a value of 2 is seen for the spatial resolution of 32, it would be relatively as bad as the value 0.5 in the plot.

The deficiency of the error estimate can be seen Figure 4.28. It can be seen that the deficiency in error estimates follows a similar trend for both the reconstructed autoencoder primal and the discrete primal. However, when looking at how similar the autoencoder deficiency is compared to the autoencoder in Figure 4.30, it can be seen that for higher spatial resolution, the autoencoder is not similar at all. Note that in Figure 4.30, the relative distance from a value of 1 measures how similar the values are. Therefore, when a value of 2 is seen for the spatial resolution of 32, it would be relatively as bad as the value 0.5 in this plot as well. This indicates that only the deficiency of the error estimates based on the autoencoder for the lowest spatial resolution is similar to the primal deficiency of the error estimates.

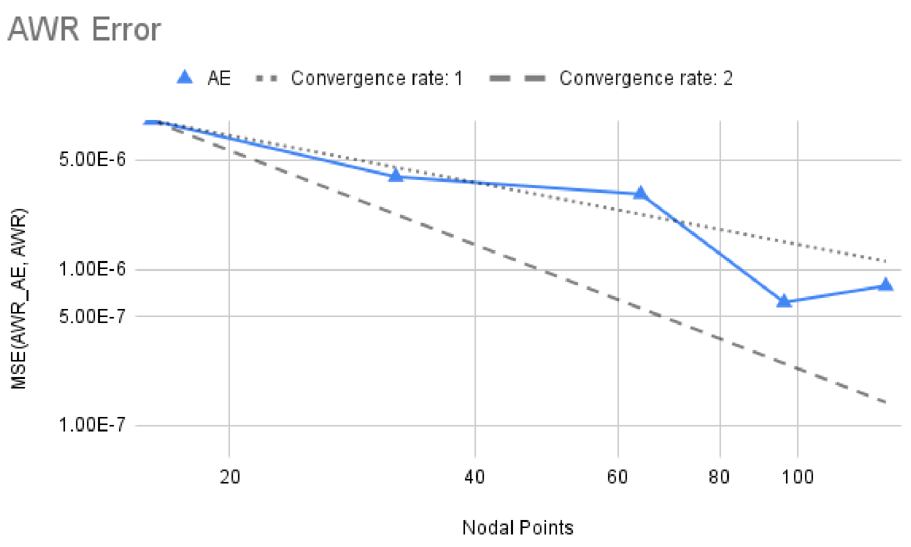

Figure 4.24: 1D Burgers MSE of the local error estimates for different spatial resolutions on the fine space

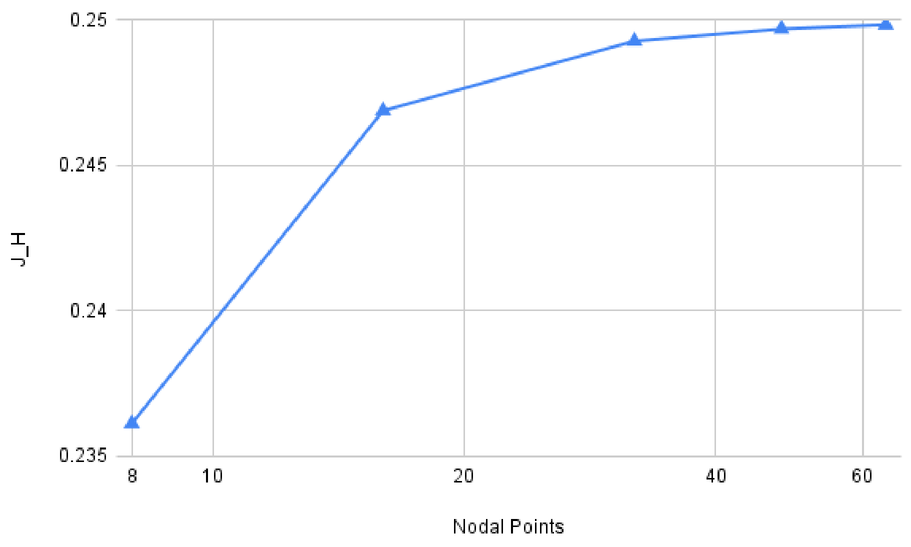

Figure 4.25: QoI values for different coarse solutions for the 1D Burgers equation

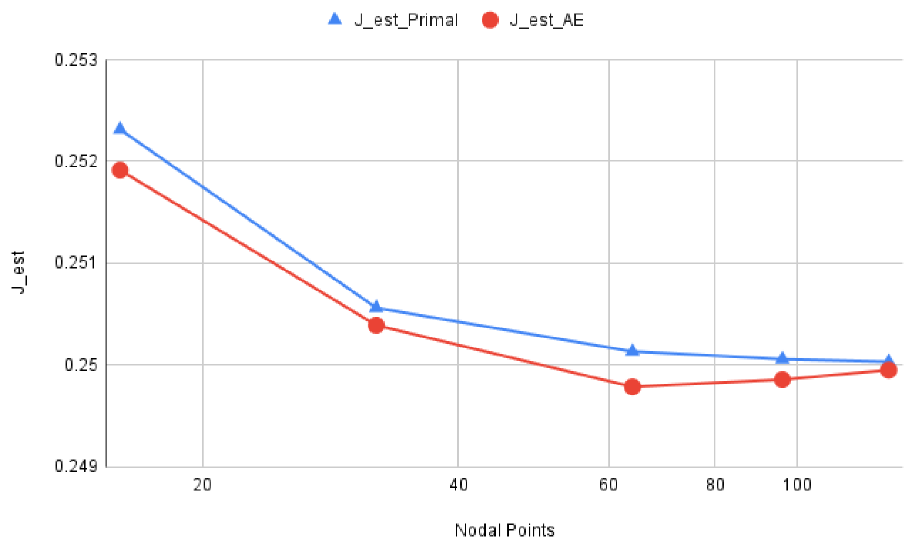

Figure 4.26: Estimated QoI on the refined space for different coarse solutions for the 1D Burgers equation

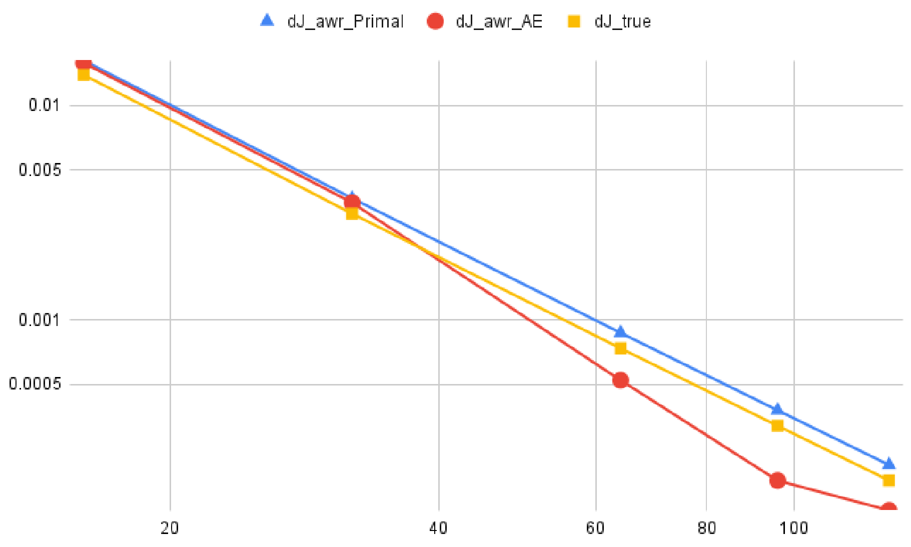

Figure 4.27: Error estimate on the refined space for different coarse solutions for the 1D Burgers equation

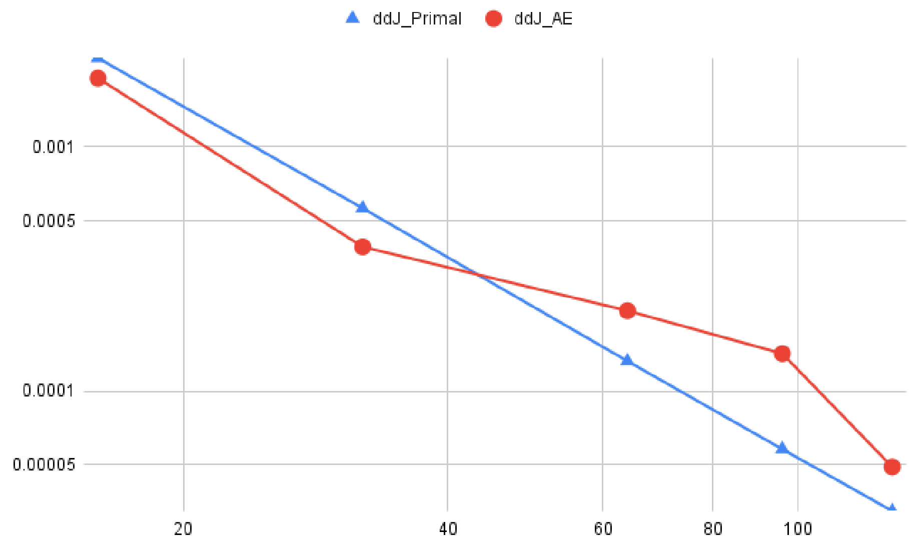

Figure 4.28: Deficiency of the error estimate on the refined space for different coarse solutions for the 1D Burgers equation

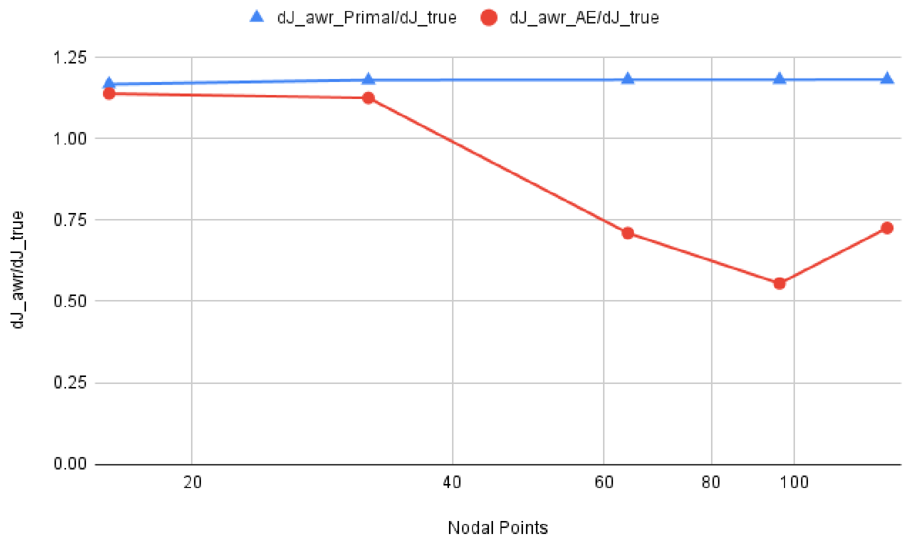

Figure 4.29:
Normalised error estimate of the autoencoder on the refined space for different coarse solutions for the 1D Burgers equation

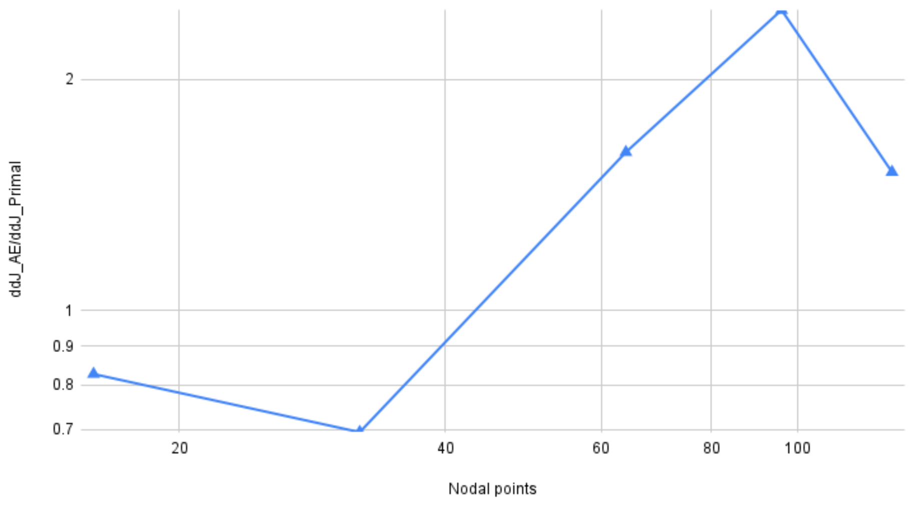

Figure 4.30: Normalised deficiency of the autoencoder error estimate on the refined space for different coarse solutions for the 1D Burgers equation

5
Lid-Driven Cavity Flow

In this chapter, the Navier-Stokes equations for the lid-driven cavity flow problem will be compressed and reconstructed. This will be done using the POD and an autoencoder neural network. The adjoint-based error estimation method will be applied to both reconstructed solutions and will be compared for similar compression ratios. The results presented in this chapter are for the resolution of 8×8, 16×16, 32×32 and 64×64 points and for the Reynolds number of 500. To start this chapter, the governing equations are presented in Section 5.1 as well as the solution obtained from these equations. Since no exact solution exists for the Navier-Stokes equations, a reference solution is generated for the primal in Section 5.2. After this POD is applied to the solution in Section 5.3. Next, the autoencoder will be applied to the solution in Section 5.4. After this, the adjoint equations will be presented and solved in Section 5.5. Lastly, the error estimates are shown in Section 5.6.

5.1. Primal Fluid Equations and Solution for lid-driven cavity flow

The lid-driven cavity flow problem is a well-known incompressible flow problem frequently used to verify and validate novel CFD methods [49, 50]. The problem in itself is a flow confined in a two-dimensional unit square region with three stationary borders and one moving top barrier (a lid). The governing equations for the lid-driven cavity flow can be seen in Equation 5.1. The unit square domain is indicated with Ω. The moving top boundary is denoted by $\partial\Omega_{\text{lid}}$ and the three remaining stationary boundaries are denoted by $\partial\Omega_{\text{wall}}$.

$$
\begin{aligned}
&\text{PDEs:} && \text{Momentum: } \frac{\partial \mathbf{u}}{\partial t}+(\mathbf{u}\cdot\nabla)\mathbf{u}=-\nabla p+\frac{1}{Re}\nabla^2\mathbf{u} \quad \text{in} \quad \Omega,\\
& && \text{Mass: } \nabla\cdot\mathbf{u}=0 \quad \text{in} \quad \Omega,\\
&\text{BCs:} && \mathbf{u}(x,y,t)=\begin{bmatrix}u_{\text{wall}}\\ v_{\text{wall}}\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix} \quad \text{on } \partial\Omega_{\text{wall}},\\
& && \mathbf{u}(x,y,t)=\begin{bmatrix}u_{\text{lid}}\\ v_{\text{lid}}\end{bmatrix}=\begin{bmatrix}1\\0\end{bmatrix} \quad \text{on } \partial\Omega_{\text{lid}},\\
& && p(0,0,t)=0\\
&\text{ICs:} && \mathbf{u}(x,y,0)=p(x,y,t)=0 \quad \text{in} \quad \Omega
\end{aligned}
\tag{5.1}
$$

The QoI for this case is $\bar{J}=\frac{1}{T}\int_{\Omega\times I}\mathbf{u}\,d\Omega dT$. This results in the source terms for the adjoint equations being $\frac{dJ}{dp}=0$ and $\frac{dJ}{d\mathbf{u}}=[1,1]^T$. Since the main region of interest is the transient phase of the flow solution, the equations will be solved for the time domain $I:[0,10]$ with a timestep of 0.01. As there is no exact solution for this QoI, a reference solution will be calculated for a more refined spatial resolution and deemed to be the most accurate representation of the solution of the Navier Stokes equation. The reference solution can be found in Section 5.2.

The primal fields for the resolution N of 8×8, 16×16, 32×32 and $64\times$ points for the Reynolds number 500 can be seen in Figure 5.2, Figure 5.2, Figure 5.3, Figure 5.4, Figure 5.5, Figure 5.6, Figure 5.7 and Figure 5.8. The figures are shown for the last time step. The primal solutions for all other time steps can be found in Appendix C.

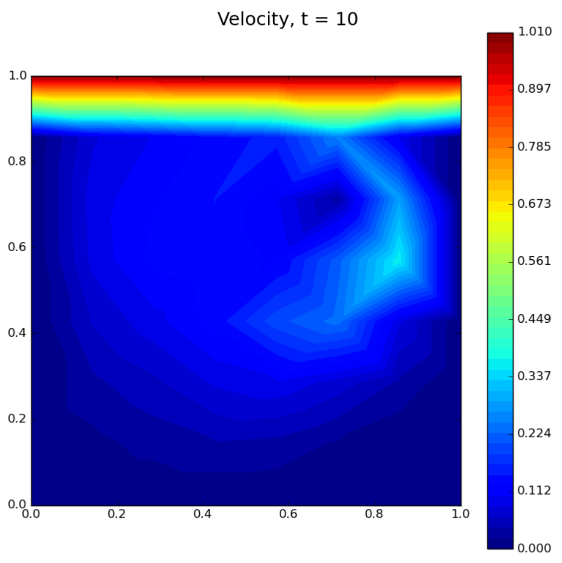

Figure 5.1: Velocity
field for $Re = 500$ and $N_H = 8 \times 8$ for the lid-driven cavity flow

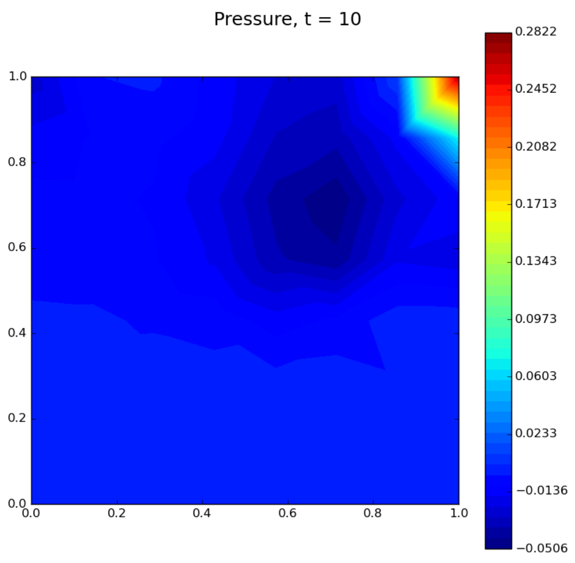

Figure 5.2: Pressure
field for $Re = 500$ and $N_H = 8 \times 8$ for the lid-driven cavity flow

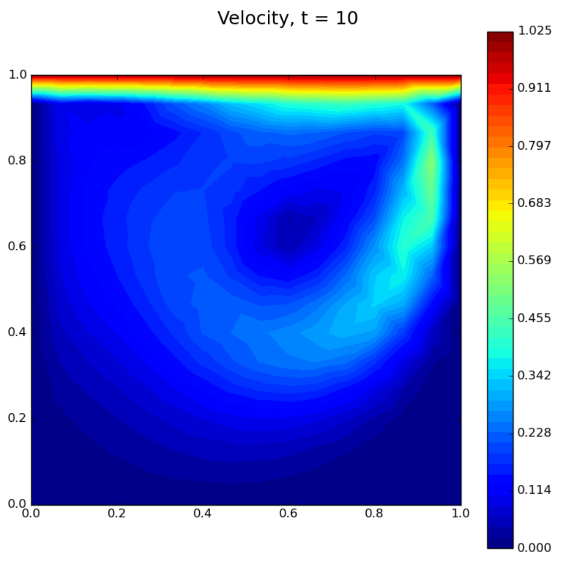

Figure 5.3: Velocity
field for $Re = 500$ and $N_H = 16 \times 16$ for the lid-driven cavity flow

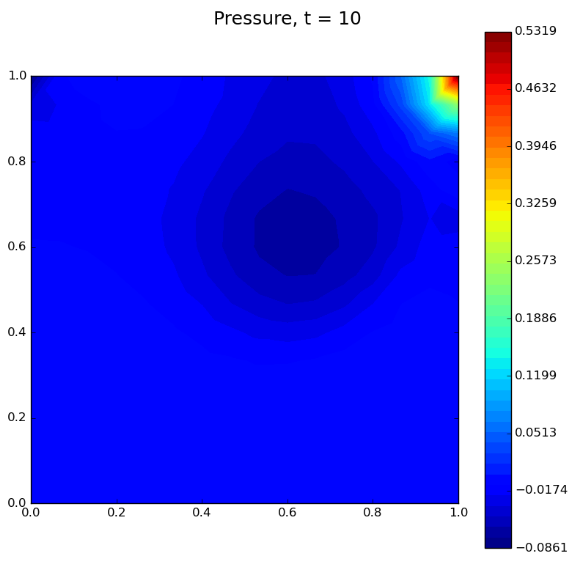

Figure 5.4: Pressure
field for $Re = 500$ and $N_H = 16 \times 16$ for the lid-driven cavity flow

5.2. Reference Solution for lid-driven cavity flow

Since no exact solution is known for the Navier-Stokes equations shown in Equation 5.1, a reference solution will be generated on a finer discretised domain. The solutions obtained are assumed to contain the least error and are closest to the true solutions of the primal solution. The reference solutions have a spatial resolution of 96×96 can be seen in Figure 5.9 and Figure 5.10. When comparing to the velocity fields results shown in [49], the results are consistent. The other time steps of the reference solution can be found in Appendix C.

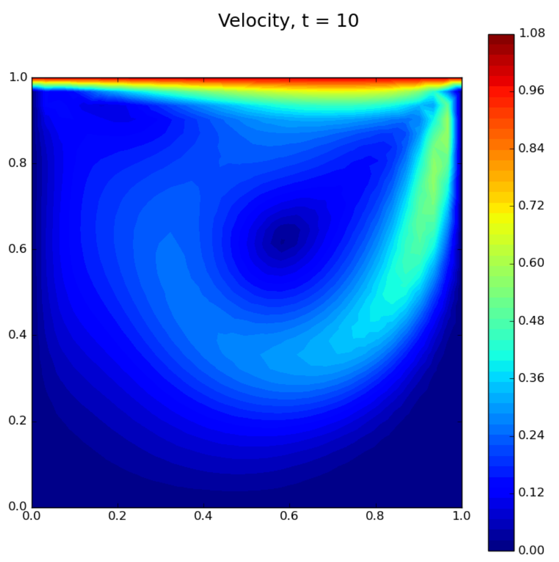

Figure 5.5: Velocity field for $Re = 500$ and $N_H = 32 \times 32$ for the lid-driven cavity flow

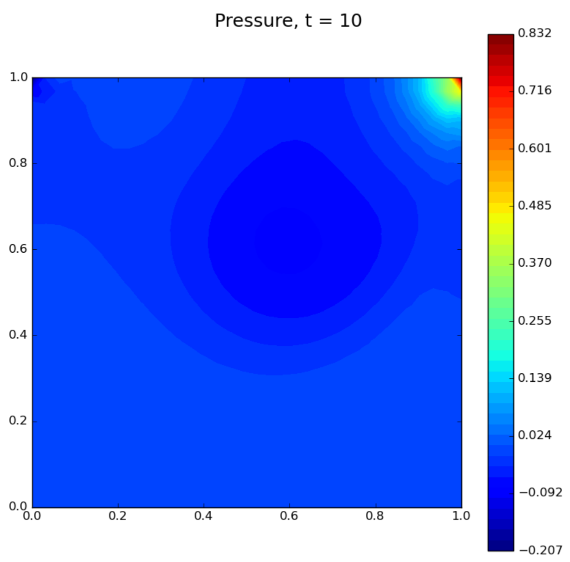

Figure 5.6: Pressure field for $Re = 500$ and $N_H = 32 \times 32$ for the lid-driven cavity flow

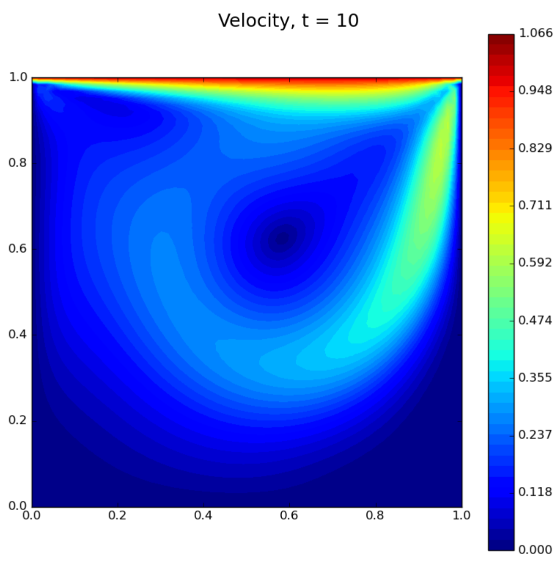

Figure 5.7: Velocity field for $Re = 500$ and $N_H = 64 \times 64$ for the lid-driven cavity flow

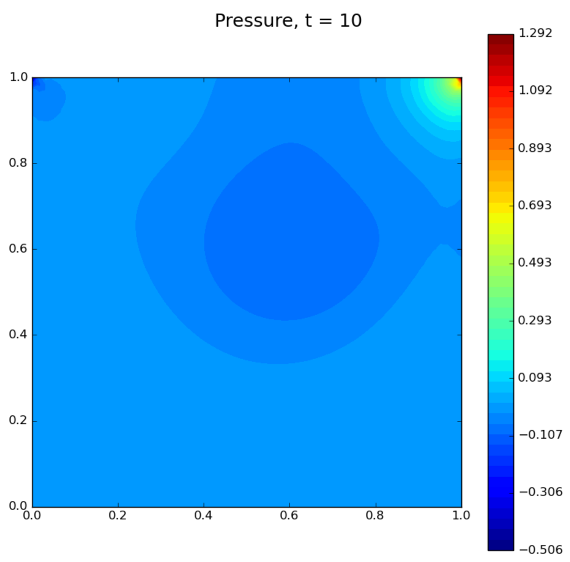

Figure 5.8: Pressure field for $Re = 500$ and $N_H = 64 \times 64$ for the lid-driven cavity flow

5.3. Compression and reconstruction using POD for lid-driven cavity flow

Since the primal solution is known, the primal solution can be compressed using the POD approach, explained in Section 3.1. Since the lid-driven cavity flow problem has three variables namely, u, v, and p, the POD approach will be applied such that 99.99% of the total mode energy is retained for all variables compressed. The POD is applied to the flow field solution. For the different resolutions, the number of modes r and the achieved compression ratio CR_{POD} can be seen in Table 5.1. The compression ratio for POD for the lid-driven cavity flow problem CR_{POD} is defined by Equation 5.2.

The mode energy ratio for all resolutions for u, v and p can be seen in Figure 5.11, Figure 5.12, Figure 5.13 and Figure 5.14. From these figures can be seen that most of the energy is contained for the first 6 modes. After this, the mode energy retained by the POD is very low. This indicates that the discrete

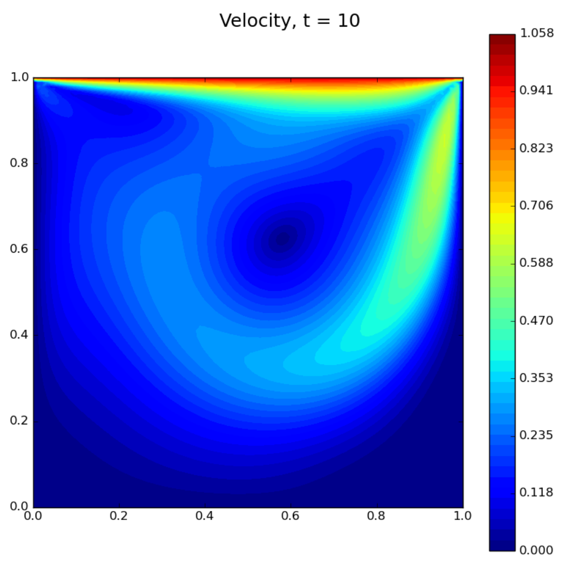

Figure 5.9: Reference velocity field for $Re = 500$ and $N_H = 96 \times 96$ for the lid-driven cavity flow

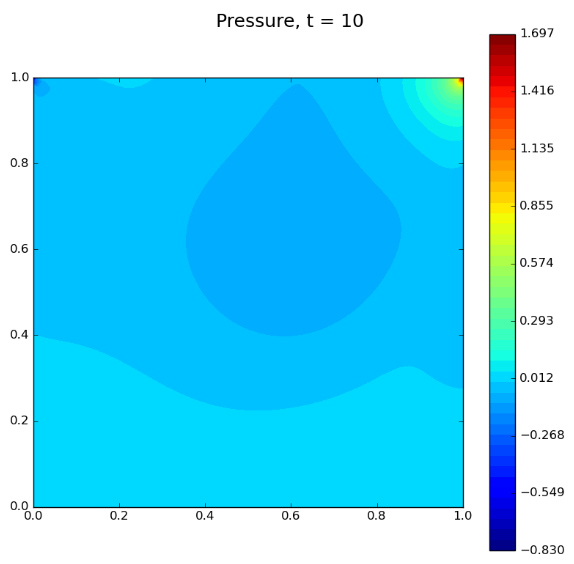

Figure 5.10: Reference pressure field for $Re = 500$ and $N_H = 96 \times 96$ for the lid-driven cavity flow

primal solution is smooth.

$$CR_{\text{POD}} = \frac{\text{Input data size}}{\text{POD output data size}} = \frac{\text{Time resolution} \cdot (\text{Spatial resolution of input})}{\text{Size}(\mathbf{U}_r) + \text{Size}(\boldsymbol{\Sigma}_r) + \text{Size}(\mathbf{V}_r^T)} \tag{5.2}$$

Spatial resolution	Number of modes r	Compression Ratio CR_{POD}
8×8	8	7.512195122
16×16	8	25.46263911
32×32	8	63.24185587
64×64	9	89.36175406

Table 5.1: POD compression information for different resolutions for a Reynolds number of 500 for the lid-driven cavity flow

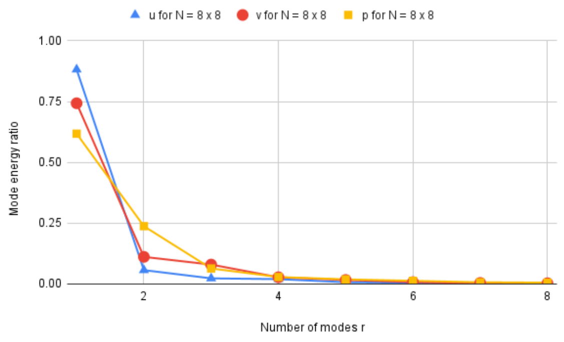

Figure 5.11: Mode energy ratio for each POD mode for $Re = 500$ and $N_H = 8 \times 8$ for the lid-driven cavity flow

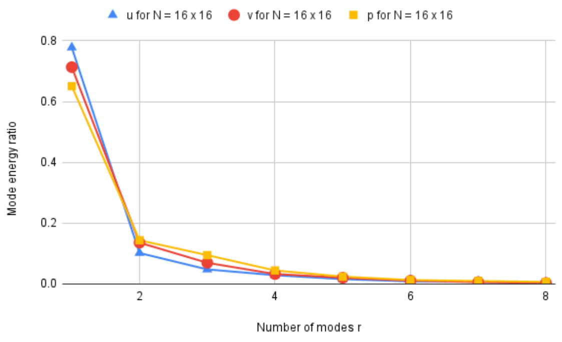

Figure 5.12: Mode energy ratio for each POD mode for $Re = 500$ and $N_H = 16 \times 16$ for the lid-driven cavity flow

The POD reconstructed velocity and pressure fields for the final time step and all resolutions can be seen in Figure 5.15, Figure 5.16, Figure 5.18, Figure 5.19, Figure 5.21, Figure 5.22, Figure 5.24 and Figure 5.25. The POD reconstruction for all other time steps can be found in Appendix C. When looking at the number of POD modes needed to retain 99.99% of the total modal energy, it can be seen that only a few modes are needed. Due to the smoothness of the discrete primal solutions, which is seen from the quick

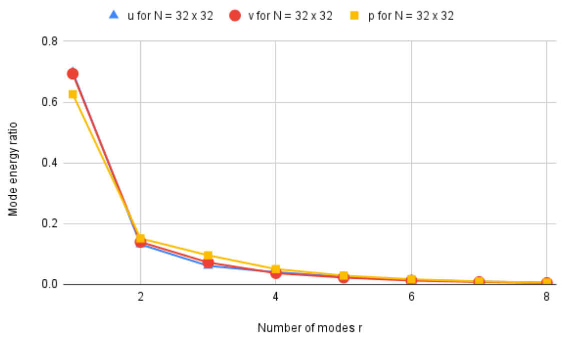

Figure 5.13: Mode energy ratio for each POD mode for $Re = 500$ and $N_H = 32 \times 32$ for the lid-driven cavity flow

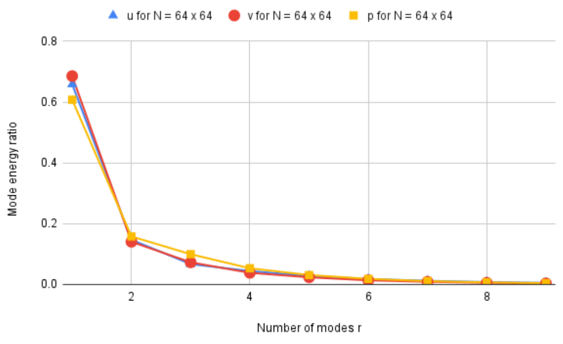

Figure 5.14: Mode energy ratio for each POD mode for $Re = 500$ and $N_H = 64 \times 64$ for the lid-driven cavity flow

convergence of the mode energy in Figure 5.11, Figure 5.12, Figure 5.13 and Figure 5.14, the POD is able to capture the solution using a low number of modes. This is also be seen in Figure 5.17, Figure 5.20, Figure 5.23 and Figure 5.26. From these figures, it can also be seen that the POD makes fewer errors as spatial resolution increases. This is due to the increase of the size of the $\mathbf{U}_r$ matrix with the spatial resolution.

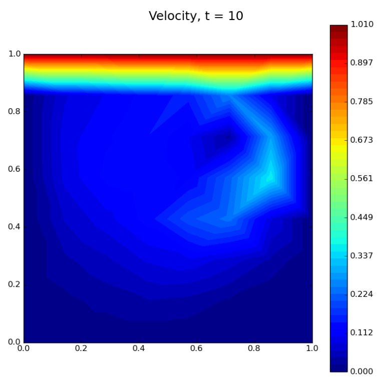

Figure 5.15: POD reconstructed velocity field for $Re = 500$ and $N_H = 8 \times 8$ for the lid-driven cavity flow

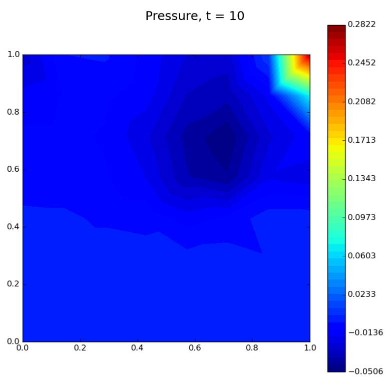

Figure 5.16: POD reconstructed pressure field for $Re = 500$ and $N_H = 8 \times 8$ for the lid-driven cavity flow

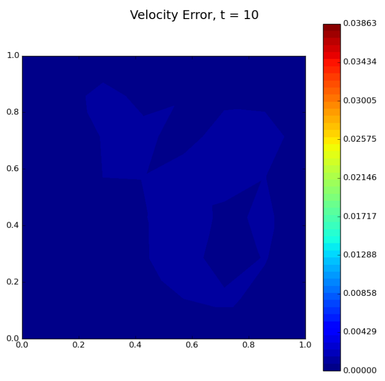

Figure 5.17: Absolute error between POD reconstructed velocity field and discrete primal velocity field for $Re = 500$ and $N_H = 8 \times 8$ for the lid-driven cavity flow

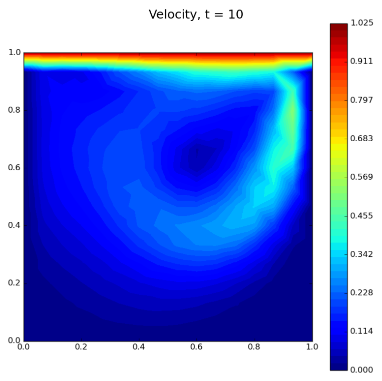

Figure 5.18: POD reconstructed velocity field for $Re = 500$ and $N_H = 16 \times 16$ for the lid-driven cavity flow

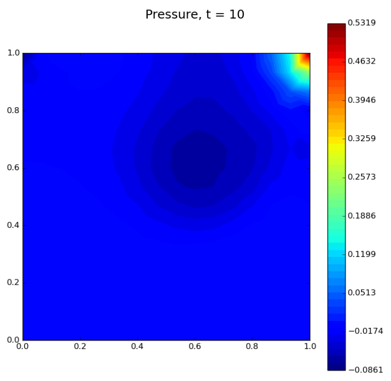

Figure 5.19: POD reconstructed pressure field for $Re = 500$ and $N_H = 16 \times 16$ for the lid-driven cavity flow

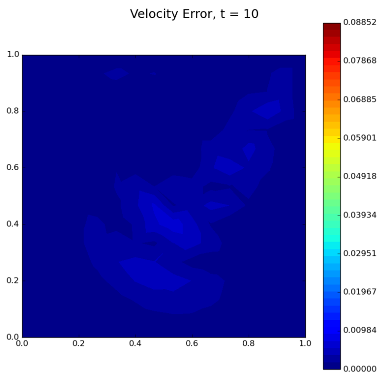

Figure 5.20: Absolute error between POD reconstructed velocity field and discrete primal velocity field for $Re = 500$ and $N_H = 16 \times 16$ for the lid-driven cavity flow

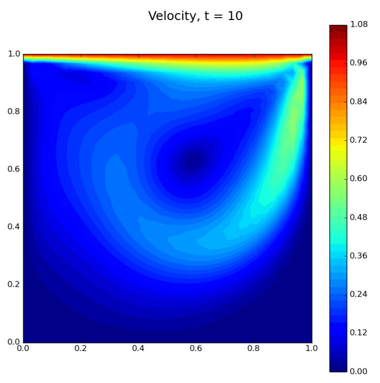
Figure 5.21: POD reconstructed velocity field for $Re = 500$ and $N_H = 32 \times 32$ for the lid-driven cavity flow

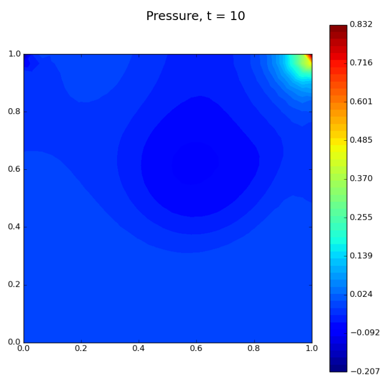
Figure 5.22: POD reconstructed pressure field for $Re = 500$ and $N_H = 32 \times 32$ for the lid-driven cavity flow

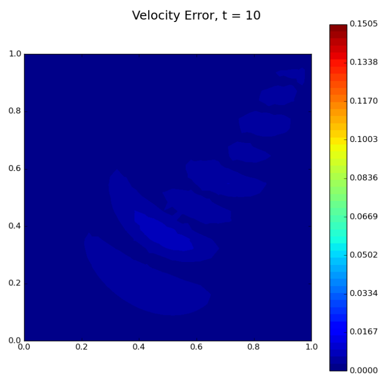
Figure 5.23: Absolute error between POD reconstructed velocity field and discrete primal velocity field for $Re = 500$ and $N_H = 32 \times 32$ for the lid-driven cavity flow

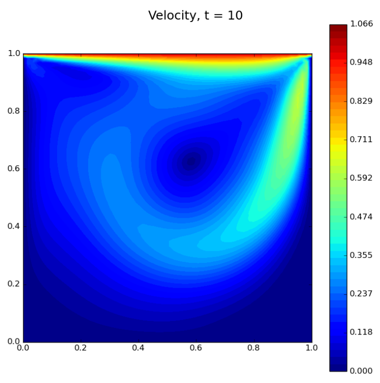
Figure 5.24: POD reconstructed velocity field for $Re = 500$ and $N_H = 64 \times 64$ for the lid-driven cavity flow

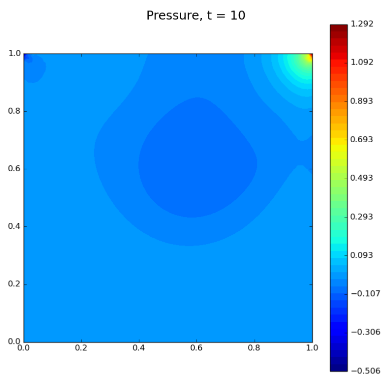
Figure 5.25: POD reconstructed pressure field for $Re = 500$ and $N_H = 64 \times 64$ for the lid-driven cavity flow

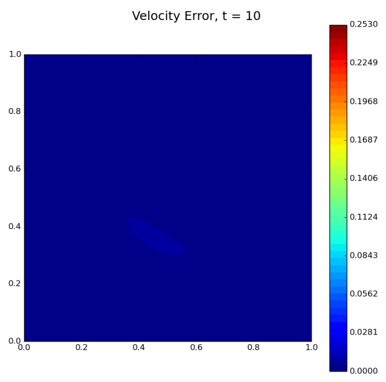
Figure 5.26: Absolute error between POD reconstructed velocity field and discrete primal velocity field for $Re = 500$ and $N_H = 64 \times 64$ for the lid-driven cavity flow

In Figure 5.27, the MSE of the reconstructed POD primal is shown. It can be seen that as the resolution is increased, the MSE gets higher until it stabilizes to a value of $1e-6$. This is due to the criterion the POD uses. As resolution is increased, the criterion does not change. When retaining 99.99% of the mode energy, the cells are allowed to have a reconstruction error. If the number of cells increases, the error increases as well. Additionally, the MSE can be seen compared to the compression ratio in Figure 5.28. The behaviour seen in Figure 5.27 is also displayed in Figure 5.28.

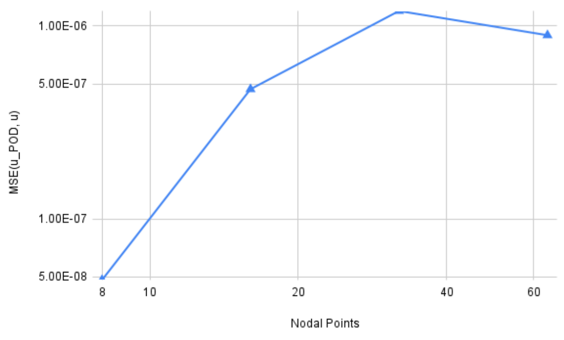

Figure 5.27: Lid-driven cavity flow MSE of the reconstructed POD primal for different resolutions

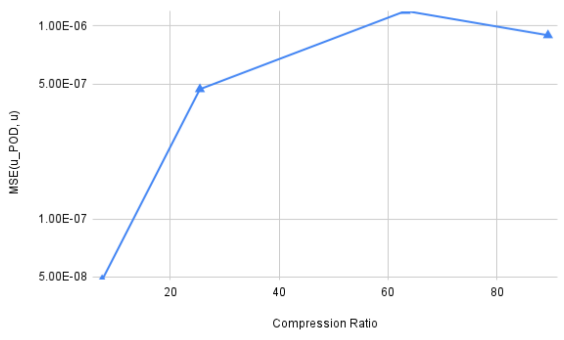

Figure 5.28: Lid-driven cavity flow MSE of the reconstructed POD primal for different compression ratios

The cost of the POD cannot be neglected when reconstructing the discrete primal solution. For this case, the POD can simply be done using a SVD. However, when moving to three-dimensional cases, the cost of solving such eigenvalue problems will become a bottleneck to the method. The only alternative to be used in practice is the EOA based on incremental SVD proposed in [17].

5.4. Compression and reconstruction using NN for lid-driven cavity flow

Now, the primal is also compressed using an autoencoder. The autoencoder is constructed using the method given in Section 3.2. As noted before, the autoencoder has a different architecture due to the fully connected linear layers. The autoencoder architecture can be seen in Appendix A. To achieve a similar compression ratio with the autoencoder as the POD, some space needs to be reserved for the autoencoder model. To do so, the size of the compressed state will be r, such that the remaining values can be used to save the model.

For the different resolutions the number of model parameters and the achieved compression ratio CR_{AE} can be seen in Table 5.2. CR_{AE} is defined by Equation 5.3. It should be noted that more model parameters will increase the computational cost of training and running the autoencoder. Even though the computational cost is not the scope of this thesis, it should not be overlooked.

$$\begin{aligned} CR_{\text{AE}} &= \frac{\text{Input data size}}{\text{Encoder output data size} + \text{Model size}} \\ &= \frac{\text{Number of variables} \times \text{Time resolution} \times \text{Spatial resolution of input}}{\text{Time resolution} \times \text{Spatial resolution of encoder output} + \text{Number of Model parameters}} \end{aligned} \tag{5.3}$$

Resolution	Model parameters	Compression Ratio CR_{AE}
8×8	17430	7.555310952
16×16	22239	25.41633881
32×32	40720	63.106879
64×64	128675	89.33709073

Table 5.2: Autoencoder compression information for different resolutions for a Reynolds number of 500 for the lid-driven cavity flow

The train and test losses for the different resolutions can be seen in Figure 5.29, Figure 5.30, Figure 5.31 and Figure 5.32. The training is done using a learning rate of $5e-4$. The train and test losses show that an MSE of 10^{-4} is achieved for a spatial resolution of 8×8 points. For the other spatial resolutions, a slightly higher MSE is achieved as can be seen in the figures.

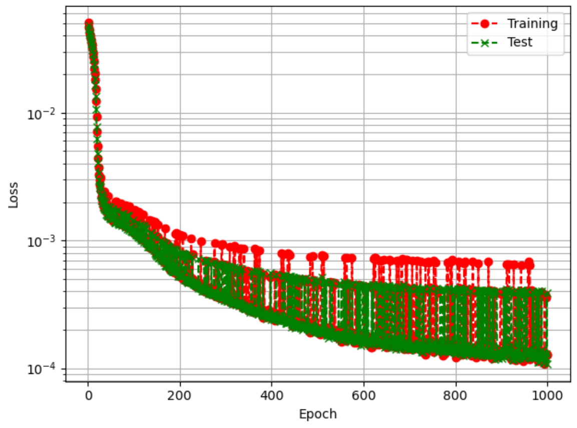

Figure 5.29: Train and test losses of the autoencoder for $Re = 500$ and $N_H = 8 \times 8$ for the lid-driven cavity flow

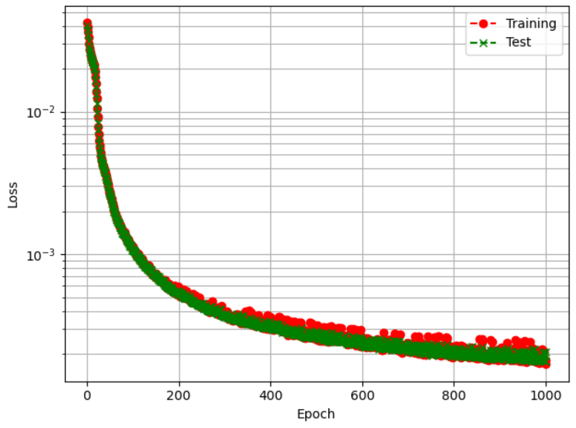

Figure 5.30: Train and test losses of the autoencoder for $Re = 500$ and $N_H = 16 \times 16$ for the lid-driven cavity flow

The autoencoder reconstructed primal fields can be seen in Figure 5.33, Figure 5.34, Figure 5.36,

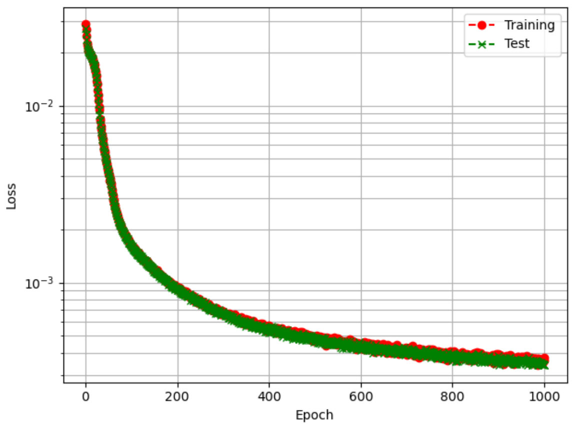

Figure 5.31: Train and test losses of the autoencoder for $Re = 500$ and $N_H = 48 \times 48$ for the lid-driven cavity flow

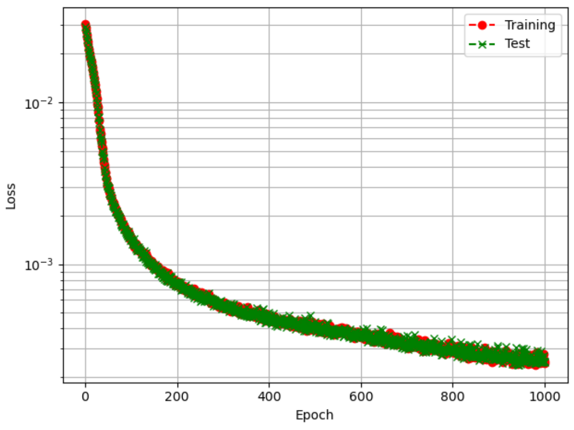

Figure 5.32: Train and test losses of the autoencoder for $Re = 500$ and $N_H = 64 \times 64$ for the lid-driven cavity flow

Figure 5.37, Figure 5.39, Figure 5.40, Figure 5.42 and Figure 5.43. It can be seen that the main features of the flow have been captured using the autoencoder neural network, however, when looking at the instantaneous solutions of the reconstructed primal for the higher resolutions, these do not look as smooth as the inputted primal solution. This indicates that the latent space for the higher resolutions is too small which results in the discrepancies seen in the autoencoder reconstructed primal solutions. For the lower resolutions, it can be seen that the autoencoder reconstruction is more accurate when compared to the discrete primal solution. In Figure 5.35, Figure 5.38, Figure 5.41 and Figure 5.44, the absolute error fields between the autoencoder reconstructed velocity and the discrete primal velocity has been shown. It can be seen that the lowest error is made for the lowest resolution and the highest errors are found for the resolution of 32×32 points. The resolution of 64×64 has a lower error due to the latent space being one size larger, therefore the absolute errors are lower.

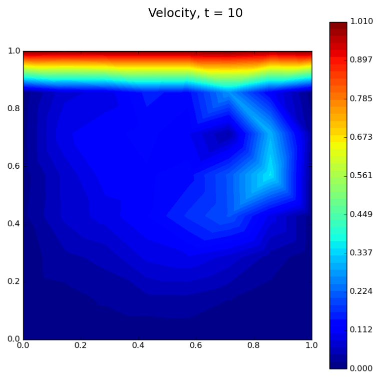

Figure 5.33: Autoencoder reconstructed velocity field for $Re = 500$ and $N_H = 8 \times 8$ for the lid-driven cavity flow

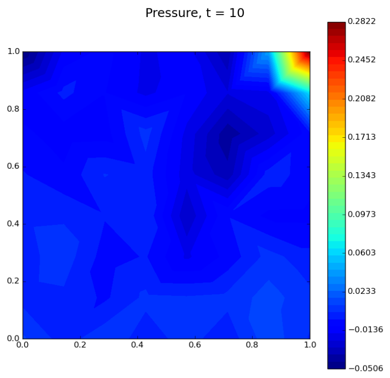

Figure 5.34: Autoencoder reconstructed pressure field for $Re = 500$ and $N_H = 8 \times 8$ for the lid-driven cavity flow

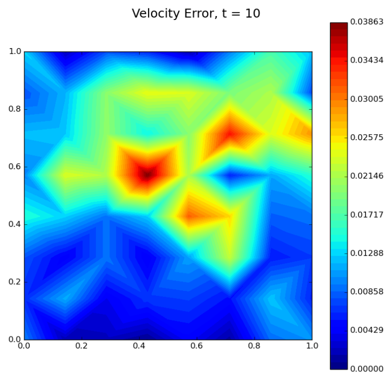

Figure 5.35: Absolute error between autoencoder reconstructed velocity field and discrete primal velocity field for $Re = 500$ and $N_H = 8 \times 8$ for the lid-driven cavity flow

In Figure 5.45, the MSE for the reconstructed autoencoder primal for different resolutions is shown. It can be seen that the MSE value for the autoencoder is two orders higher than the MSE of the POD. The reconstruction for the lower resolutions can be seen to be more accurate due to the small latent space size. The resolution of 64 shows a lower MSE due to the increase of the latent space. The MSE has also been plotted against the compression ratio in Figure 5.46, providing a mapping of the error made for each compression ratio investigated in this thesis. Note that the MSE depends on the autoencoder architecture used as well.

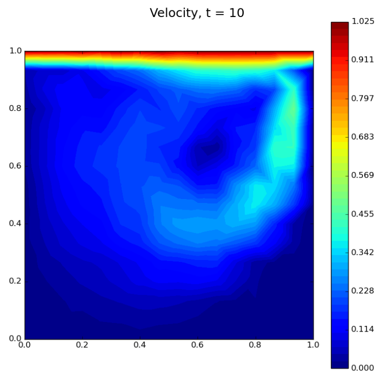

Figure 5.36: Autoencoder reconstructed velocity field for $Re = 500$ and $N_H = 16 \times 16$ for the lid-driven cavity flow

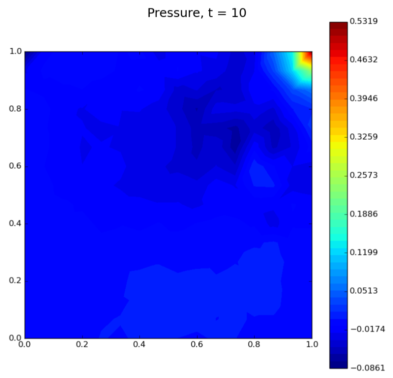

Figure 5.37: Autoencoder reconstructed pressure field for $Re = 500$ and $N_H = 16 \times 16$ for the lid-driven cavity flow

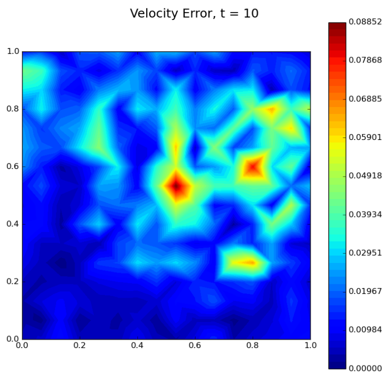

Figure 5.38: Absolute error between autoencoder reconstructed velocity field and discrete primal velocity field for $Re = 500$ and $N_H = 16 \times 16$ for the lid-driven cavity flow

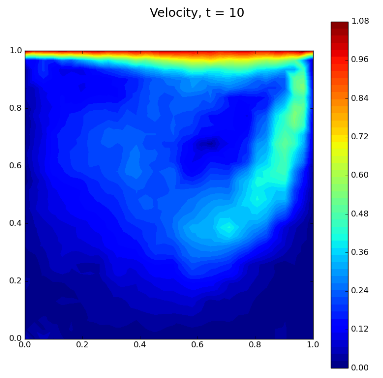

Figure 5.39: Autoencoder reconstructed velocity field for $Re = 500$ and $N_H = 32 \times 32$ for the lid-driven cavity flow

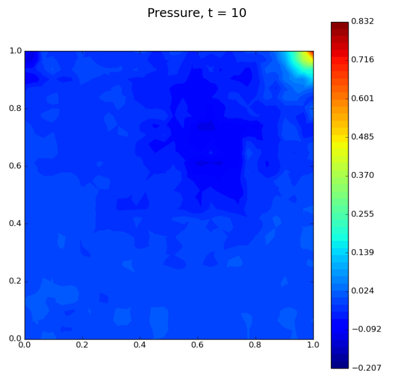

Figure 5.40: Autoencoder reconstructed pressure field for $Re = 500$ and $N_H = 32 \times 32$ for the lid-driven cavity flow

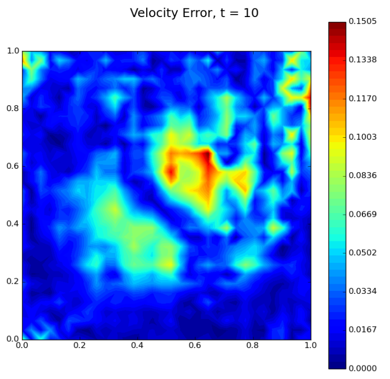

Figure 5.41: Absolute error between autoencoder reconstructed velocity field and discrete primal velocity field for $Re = 500$ and $N_H = 32 \times 32$ for the lid-driven cavity flow

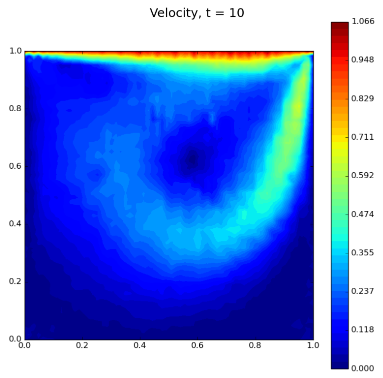

Figure 5.42: Autoencoder reconstructed velocity field for $Re = 500$ and $N_H = 64 \times 64$ for the lid-driven cavity flow

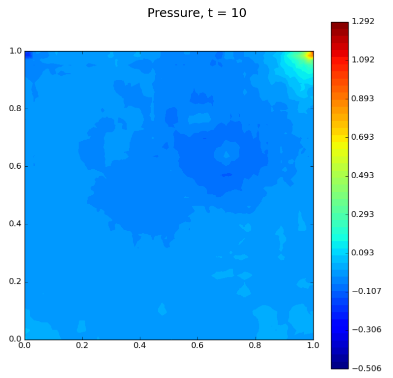

Figure 5.43: Autoencoder reconstructed pressure field for $Re = 500$ and $N_H = 64 \times 64$ for the lid-driven cavity flow

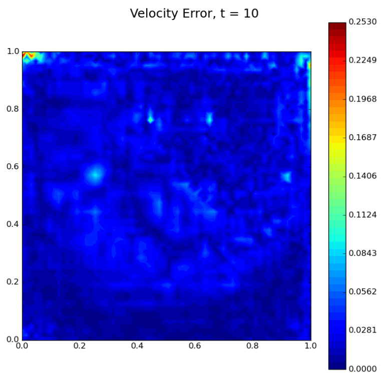

Figure 5.44: Absolute error between autoencoder reconstructed velocity field and discrete primal velocity field for $Re = 500$ and $N_H = 64 \times 64$ for the lid-driven cavity flow

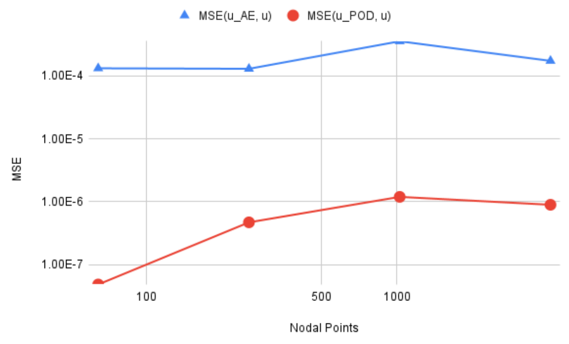

Figure 5.45: Lid-driven cavity flow MSE of the reconstructed autoencoder primal for different resolutions

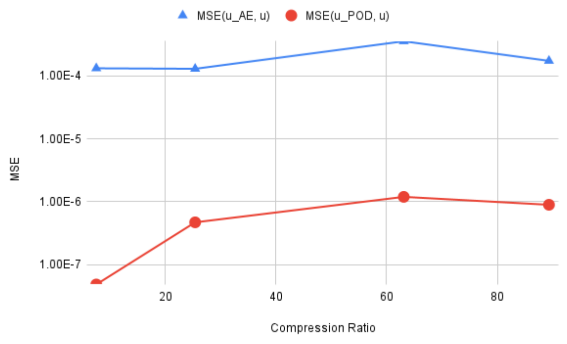

Figure 5.46: Lid-driven cavity flow MSE of the reconstructed autoencoder primal for different compression ratios

5.5. Adjoint Equations and Solution for lid-driven cavity flow

Now that $\mathbf{u}$,p,$\mathbf{u}_{\text{POD}}$,p_{POD},$\mathbf{u}_{\text{AE}}$ and p_{AE} are known, the adjoint can be calculated based on these solutions. To do so, the adjoint equation first needs to be derived. The adjoint for the linearized steady Navier Stokes equations is derived in [51]. This case is adapted for the unsteady case and can be seen in Equation 5.4. In Equation 5.4, the adjoint solution Ψ consists of the velocity adjoint $\Psi_{\mathbf{u}}$ and the pressure adjoint Ψ_p.

$$\begin{aligned}
&\text{Adjoint:} && \text{Momentum:} \quad \frac{dJ}{d\mathbf{u}} = -\frac{\partial \Psi_{\mathbf{u}}}{\partial t} - (\mathbf{u}_0 \cdot \nabla)\Psi_{\mathbf{u}} + (\Psi_{\mathbf{u}} \cdot \nabla)\mathbf{u}_0 - \nu \nabla^2 \Psi_{\mathbf{u}} - \nabla \Psi_p = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \\
& && \text{Mass:} \quad \frac{dJ}{dp} = -\nabla \cdot \Psi_{\mathbf{u}} = 0 \\
&\text{BC:} && \Psi_{\mathbf{u}}(x,y,t) = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \quad \text{on } \partial\Omega_{\text{wall}} \\
& && \nu \nabla \Psi_{\mathbf{u}} \cdot \mathbf{n} + \Psi_p \mathbf{n} + (\mathbf{u}_0 \cdot \mathbf{n})\Psi_{\mathbf{u}} = 0 \quad \text{on } \partial\Omega_{\text{lid}}, \\
& && \Psi_{\mathbf{u}} \times \mathbf{n} = 1 \quad \text{on } \partial\Omega_{\text{lid}} \\
&\text{IC:} && \Psi_{\mathbf{u}}(x,y,10) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{on } \Omega \\
& && \Psi_p(x,y,10) = 0 \quad \text{on } \Omega
\end{aligned} \tag{5.4}$$

The weak form can be derived from Equation 5.4 as well and the Crank Nicolson time scheme can be implemented for the terms containing only velocity adjoints $\Psi_{\mathbf{u}}$ and velocity test function v. The pressure adjoint Ψ_p and the pressure test function q will remain outside of this scheme as can be seen in Equation 5.5. The linearized velocities are indicated by $\mathbf{u}_i$ and $\mathbf{u}_{i-1}$. Note that the time step Δt in Equation 5.5 is a negative number as the problem flows backward in time. This also means that the problem is solved backwards in time for the velocity adjoint $\Psi_{\mathbf{u}}^{i-1}$ and the pressure adjoint Ψ_p. The brackets in Equation 5.5 represent the integrated inner product over the domain given in the subscript and the Crank-Nicolson constant θ has a value of 0.5.

$$\begin{aligned}
&\left(\frac{\Psi_{\mathbf{u}}^i - \Psi_{\mathbf{u}}^{i-1}}{\Delta t}, v\right)_\Omega + (\Psi_p \nabla, v)_\Omega - (\nabla \cdot \Psi_{\mathbf{u}}, q)_\Omega + (\Psi_p \cdot \mathbf{n}, v)_{\partial\Omega} + \\
&\theta((\nabla \Psi_{\mathbf{u}}^i \cdot \mathbf{u}_i, v)_\Omega + ((\nabla \mathbf{u}_i)^T \Psi_{\mathbf{u}}^i, v) + \nu(\nabla \Psi_{\mathbf{u}}^i, \nabla v)_\Omega - (g,v)_\Omega + \\
&\nu(\nabla \Psi_{\mathbf{u}}^i \cdot \mathbf{n}, v)_{\partial\Omega} + ((\mathbf{u}_i \cdot \mathbf{n})\Psi_{\mathbf{u}}^i, v)_{\partial\Omega}) + \\
&(1-\theta)((\nabla \Psi_{\mathbf{u}}^{i-1} \cdot \mathbf{u}_{i-1}, v)_\Omega + ((\nabla \mathbf{u}_{i-1})^T \Psi_{\mathbf{u}}^{i-1}, v) + \nu(\nabla \Psi_{\mathbf{u}}^{i-1}, \nabla v)_\Omega - (g,v)_\Omega + \\
&\nu(\nabla \Psi_{\mathbf{u}}^{i-1} \cdot \mathbf{n}, v)_{\partial\Omega} + ((\mathbf{u}_{i-1} \cdot \mathbf{n})\Psi_{\mathbf{u}}^{i-1}, v)_{\partial\Omega}) = 0
\end{aligned} \tag{5.5}$$

The adjoint solution can be seen for a resolution of 16×16, 32×32, and 64×64 for $t = 4$. All other time steps can be seen in Appendix C. For a resolution of 16×16, the adjoint solution based on the discrete primal, the reconstructed POD primal, and the autoencoder reconstructed primal is shown in Figure 5.47, Figure 5.48 and Figure 5.49, respectively. It can be seen that all three solutions qualitatively look similar. The same similarties are seen for a resolution of 32×32 in Figure 5.52, Figure 5.53 and Figure 5.54, and for a resolution of 64×64 in Figure 5.57, Figure 5.58 and Figure 5.59.

When looking at the absolute error plots for a resolution of 16×16 shown in Figure 5.50 and Figure 5.51, it can be seen that the adjoint based on the POD reconstructed primal almost has no error compared to the adjoint based on the discrete primal. This is also seen for the other two resolution in Figure 5.55 and Figure 5.60. For the autoencoder adjoint in Figure 5.51, the maximum error is seen in the top left corner, while lower errors are present around the forming cavity. The error of the top left corner is also present in the absolute error plots for a resolution of 32×32 and 64×64 in Figure 5.56 and Figure 5.61. For the resolution of 64×64, it can be seen that a large error is made using the autoencoder near the right boundary of the domain which is a result of the autoencoder reconstructed primal being less accurate for higher resolutions due to small latent space size.

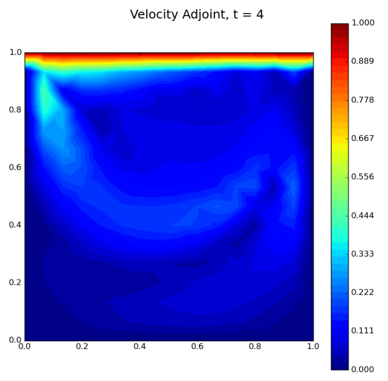

Figure 5.47: Velocity adjoint based on the original primal for $Re = 500$ and $N_h = 16 \times 16$ for the lid-driven cavity flow

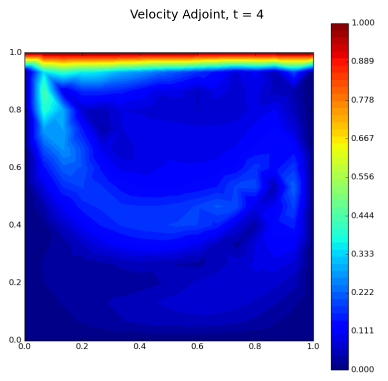

Figure 5.48: Velocity adjoint based on the POD reconstructed primal for $Re = 500$ and $N_h = 16 \times 16$ for the lid-driven cavity flow

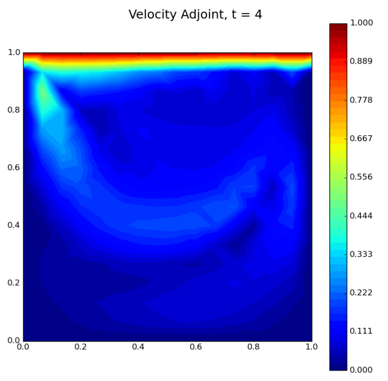

Figure 5.49: Velocity adjoint based on the autoencoder reconstructed primal for $Re = 500$ and $N_h = 16 \times 16$ for the lid-driven cavity flow

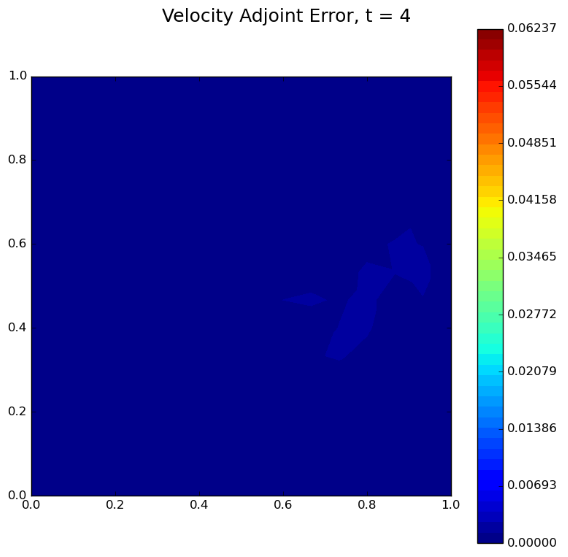

Figure 5.50: Velocity adjoint error between the discrete primal adjoint and POD-based primal adjoint for $Re = 500$ and $N_h = 16 \times 16$ for the lid-driven cavity flow

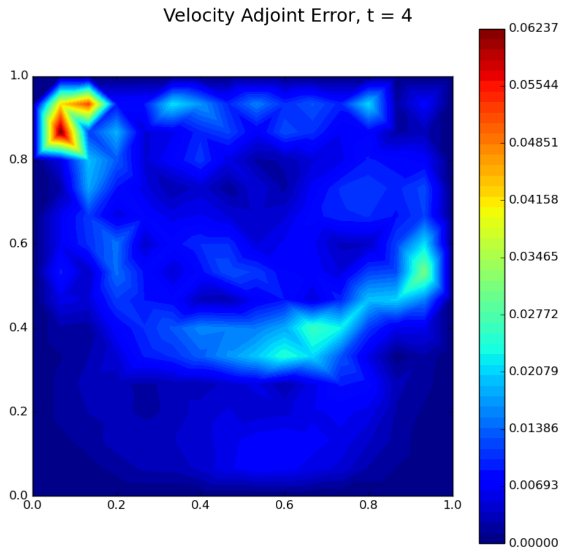

Figure 5.51: Velocity adjoint error between the discrete primal adjoint and autoencoder based primal adjoint for $Re = 500$ and $N_h = 16 \times 16$ for the lid-driven cavity flow

Finally, the MSE of the adjoint using the POD and autoencoder reconstructed primal is shown in Figure 5.62. The mapping of the compression ratio with the MSE can be seen in Figure 5.63 for the POD and the autoencoder adjoints. For higher resolutions, higher errors are seen. For the POD, this is due to the increase in resolution retaining the same level of error for all the elements. For the autoencoder, this is due to the fixed latent space size which is seen to be too small. This results in the lowest MSE being made for the lowest spatial resolution. The error increases as resolution increases while the latent space

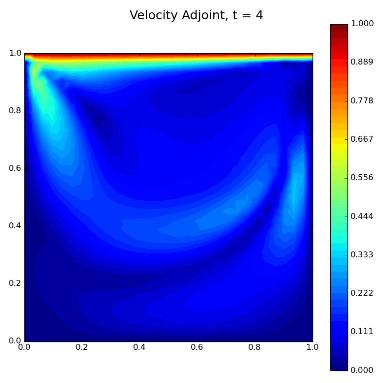

Figure 5.52: Velocity adjoint based on the original primal for $Re = 500$ and $N_h = 32 \times 32$ for the lid-driven cavity flow

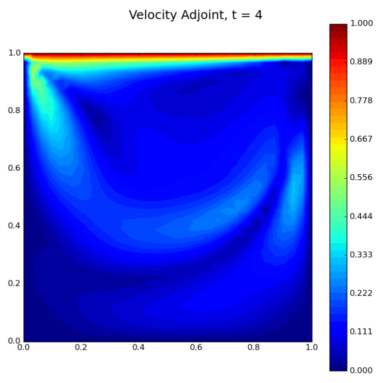

Figure 5.53: Velocity adjoint based on the POD reconstructed primal for $Re = 500$ and $N_h = 32 \times 32$ for the lid-driven cavity flow

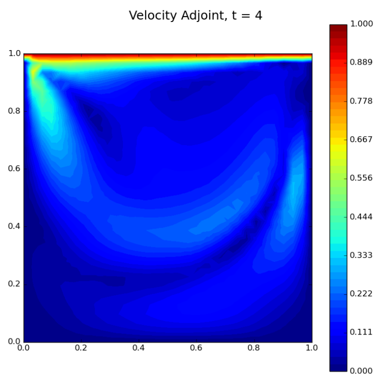

Figure 5.54: Velocity adjoint based on the autoencoder reconstructed primal for $Re = 500$ and $N_h = 32 \times 32$ for the lid-driven cavity flow

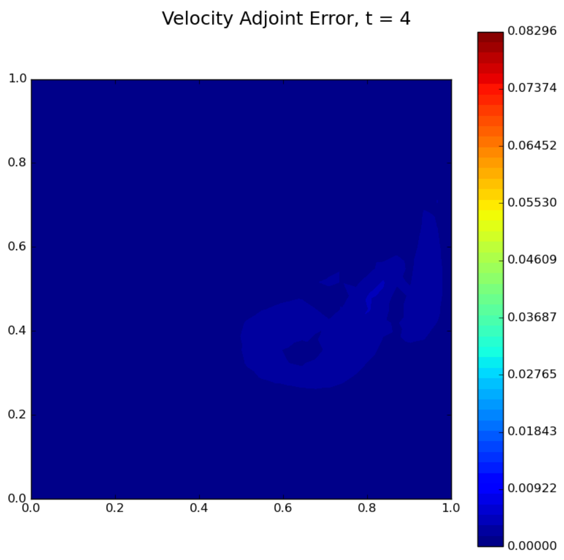

Figure 5.55: Velocity adjoint error between the discrete primal adjoint and POD-based primal adjoint for $Re = 500$ and $N_h = 32 \times 32$ for the lid-driven cavity flow

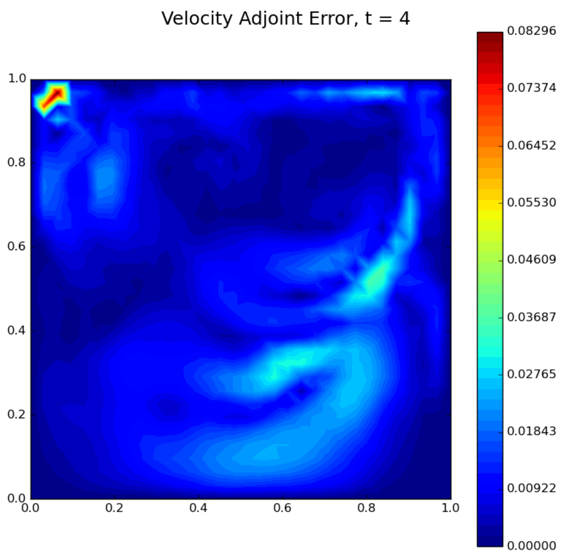

Figure 5.56: Velocity adjoint error between the discrete primal adjoint and autoencoder based primal adjoint for $Re = 500$ and $N_h = 32 \times 32$ for the lid-driven cavity flow

and number of modes remain constant. The error is seen to reduce when the latent space and number of modes is increased, as can be seen in the case for a spatial resolution of 64×64.

5.6. Adjoint-based Error Estimates for lid-driven cavity flow

To find the error estimate $\delta \bar{J}_{\text{est}}$, Equation 3.9 needs to be adapted since the adjoint equation has two adjoint solutions, namely one for the velocity and one for the pressure. To find $\delta \bar{J}_{\text{est}}$ for the Navier Stokes equations, Equation 5.6 is used. The adjoint weighted residual needs to be integrated over the spatial domain Ω and time domain I as can be seen in Equation 5.6.

$$\delta \bar{J}_{\text{est}} = -\int_I \left(\int_\Omega \Psi_{h,\mathbf{u}} \cdot \mathbf{R}_{h,\text{Mom}}(\mathbf{u}_h^H) d\Omega + \int_\Omega \Psi_{h,p} \mathbf{R}_{h,\text{Mass}}(\mathbf{u}_h^H) d\Omega \right) dT \tag{5.6}$$

To get the error estimates, all variables given in Equation 5.6 should be defined on the fine space h which is two times as refined as the coarse space H. Since all primal solutions obtained thus far are defined on the coarse space H, the primal solutions need to be projected in the fine space h which is done using interpolation in space. This yields $\mathbf{u}_h^H$, $\mathbf{u}_{\text{POD},h}^H$ and $\mathbf{u}_{\text{AE},h}^H$ for the velocity and p_h^H, $p_{\text{POD},h}^H$ and

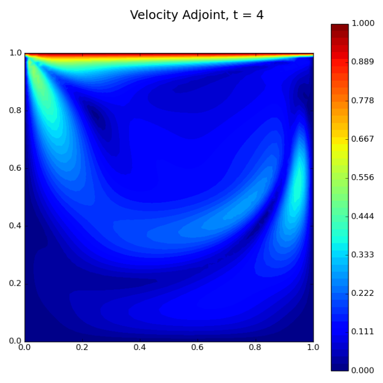

Figure 5.57: Velocity adjoint based on the original primal for $Re = 500$ and $N_h = 64\times 64$ for the lid-driven cavity flow

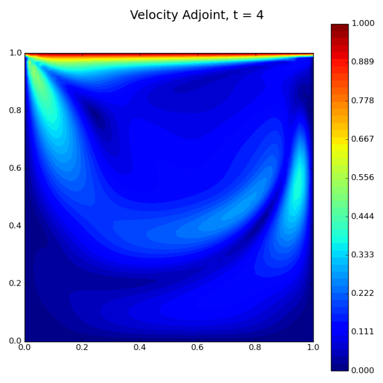

Figure 5.58: Velocity adjoint based on the POD reconstructed primal for $Re = 500$ and $N_h = 64\times 64$ for the lid-driven cavity flow

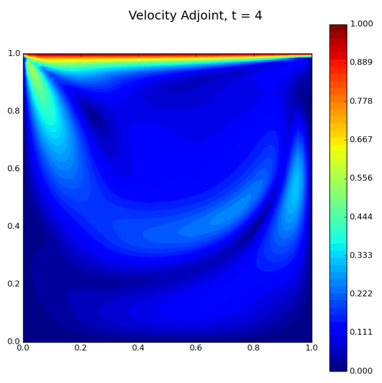

Figure 5.59: Velocity adjoint based on the autoencoder reconstructed primal for $Re = 500$ and $N_h = 64\times 64$ for the lid-driven cavity flow

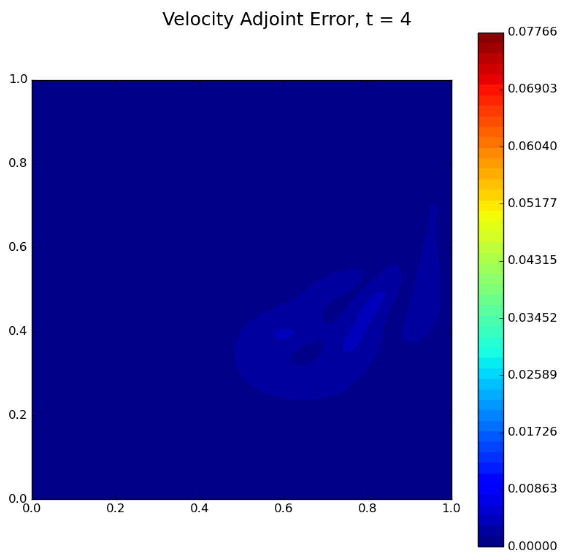

Figure 5.60: Velocity adjoint error between the discrete primal adjoint and POD-based primal adjoint for $Re = 500$ and $N_h = 64\times 64$ for the lid-driven cavity flow

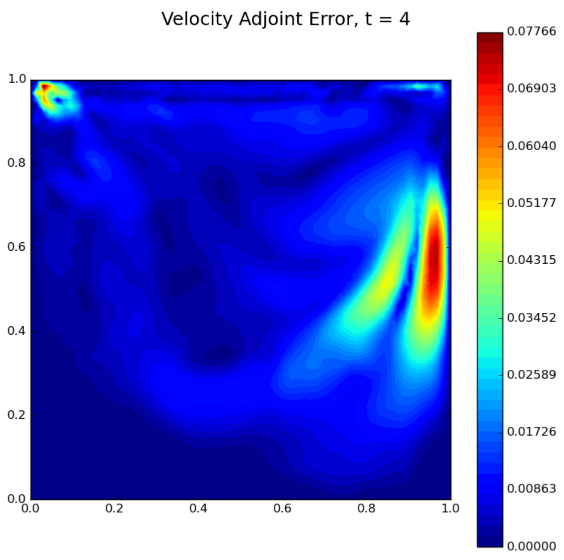

Figure 5.61: Velocity adjoint error between the discrete primal adjoint and autoencoder based primal adjoint for $Re = 500$ and $N_h = 64\times 64$ for the lid-driven cavity flow

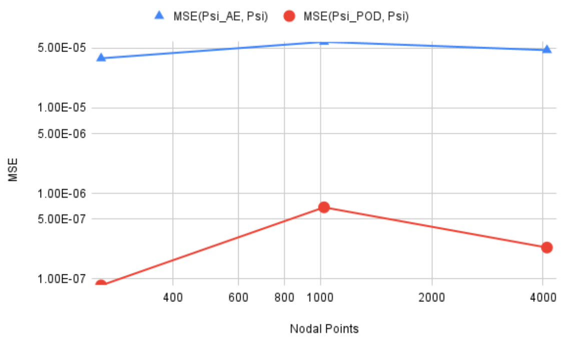

Figure 5.62: Adjoint MSE of the POD reconstructed primal for different resolutions for the lid-driven cavity flow

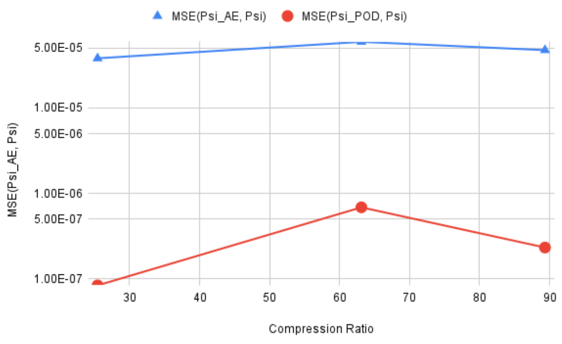

Figure 5.63: Adjoint MSE of the POD reconstructed primal for different compression ratios for the lid-driven cavity flow

$p^H_{\mathrm{AE},h}$ which will be used in the error estimation method.

The residual of the primal for both the mass and momentum equation on the fine space is given by Equation 5.7 which is derived from Equation 5.1.

$$\begin{aligned} \text{Momentum:} \quad & \mathbf{R}_{h,\mathrm{Mom}}(\mathbf{u}^H_h) = \frac{\partial \mathbf{u}^H_h}{\partial t} + (\mathbf{u}^H_h \cdot \nabla)\mathbf{u}^H_h + \nabla p^H_h - \frac{1}{Re}\nabla^2 \mathbf{u}^H_h \\ \text{Mass:} \quad & \mathbf{R}_{h,\mathrm{Mass}}(\mathbf{u}^H_h) = \nabla \cdot \mathbf{u}^H_h \end{aligned} \tag{5.7}$$

The adjoint solutions $\Psi_{\mathbf{u}}$ and Ψ_p which are obtained from Section 5.5 have been calculated on the fine space h.

The values of $\bar{J_H}$ can be computed as well using the definition of the QoI. Note that $\bar{J_{\mathrm{est}}}$ can also be calculated using the adjoint-based error estimates adapting Equation 3.8 to $\bar{J_{\mathrm{est}}} = \bar{J_H} + \delta \bar{J_{\mathrm{est}}}$.

The localized error estimate on the fine space is retrieved from Equation 5.8. This can be seen in Figure 5.64, Figure 5.65, and Figure 5.66 for a resolution of 16×16. For these figures, a log scale is used to emphasize the differences between the small values. When comparing the local estimate plots shown in Figure 5.64, Figure 5.65, and Figure 5.66, little difference is seen. When looking at the absolute error plots shown in Figure 5.67 and Figure 5.68, it can be seen that the POD adjoint weighted residual is very accurate. The autoencoder does show higher errors in the absolute value however, despite these errors, the adjoint weighted residual shown in Figure 5.66 is still accurate and has all the dominant features shown in Figure 5.64.

For the resolution 32×32 and 64×64 the local error estimates can be seen in Figure 5.69, Figure 5.70, and Figure 5.71 and Figure 5.74, Figure 5.75, and Figure 5.76 for the discrete primal, the reconstructed POD primal and the autoencoder reconstructed primal, respectively. Looking at the three figures for each resolution it can already be seen in the 32×32 resolution case that the autoencoder adjoint weighted residual is off. For the resolution of 64×64, the adjoint weighted residual is even more off. This is due to the reconstruction of the autoencoder being less accurate for higher resolution. Since the residual is used to compute the local error estimates, the autoencoder also needs to be able to reconstruct a primal that accurately captures the residual. This is more demanding for the autoencoder to do due to the residual being more complex for higher resolutions. This is why larger differences are seen for higher resolutions as is displayed in the absolute error plots shown in Figure 5.73 and Figure 5.78. The adjoint weighted residual based on the POD reconstructed primal can be seen to be accurate not only qualitatively, but also by the absolute error plots shown in Figure 5.72 and Figure 5.77.

When the residual is captured accurately enough, meaning that a large enough latent space should be used for the autoencoder, the adjoint weighted residual based on the autoencoder could be used as an error indicator for mesh refinement. It should be noted that after the refinement has been done, an even larger latent space for the autoencoder should be used which results in constructing an entirely new autoencoder after each iteration.

$$\varepsilon_{est} = \left| (\Psi^T_{h,\mathbf{u}})\mathbf{R}_{h,\mathrm{Mom}}\left(\mathbf{u}^H_h\right) + (\Psi_{h,p})\mathbf{R}_{h,\mathrm{Mass}}(\mathbf{u}^H_h) \right| \tag{5.8}$$

The QoI based on the coarse solution $\bar{J_H}$ can be seen in Figure 5.79. It can be seen that the QoI tends towards zero as the resolution increases.

The adjoint-based error estimate using the discrete primal, the POD reconstructed primal and the autoencoder reconstructed primal is shown in Figure 5.80. It can be seen that the POD reconstructed and the discrete primal solution have the same error estimate values for all resolutions. This is due to the POD being able to capture the solution easily due to the smoothness of the discrete primal. The autoencoder shows for the resolution of 16×16 that the error estimate is similar to the error estimate based on the discrete primal. As the resolution increases, the residual is being reconstructed less accurately resulting in a more erroneous adjoint-based error estimate. For the resolution of 32×32, it can already be seen to affect the error estimates. This effect is seen to be not large, even though the adjoint weighted residual was erroneous for this spatial resolution. The accuracy of the error estimate could be due to cancellation happening when computing the integral over space and time for the QoI. But when looking at the error estimate for a resolution of 64×64, it can be seen that the estimation of the error completely crumbles due to high reconstruction error in the residual. As a result of this behavior, the QoI estimates shown in Figure 5.81, are seen to be reasonably accurate for the resolutions of 16×16 and 32×32 but not accurate for the resolution of 64×64. This is shown more clearly in Figure 5.83, where the similarity is quantified between the true error and the error estimates. The definition of the true error is seen in

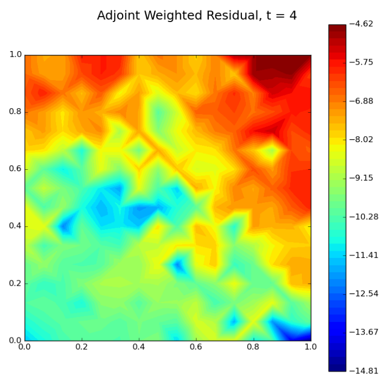

Figure 5.64: Local Error estimate field based on the original primal for $Re=500$ and $N_h=16\times16$ for the lid-driven cavity flow

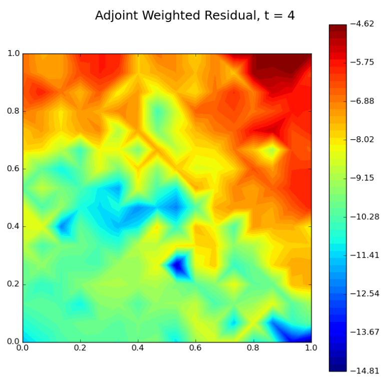

Figure 5.65: Local Error estimate field based on the original primal for $Re=500$ and $N_h=16\times16$ for the lid-driven cavity flow

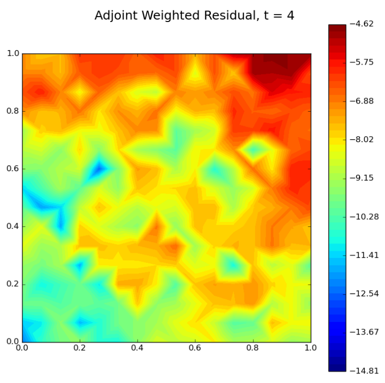

Figure 5.66: Local Error estimate field based on the original primal for $Re=500$ and $N_h=16\times16$ for the lid-driven cavity flow

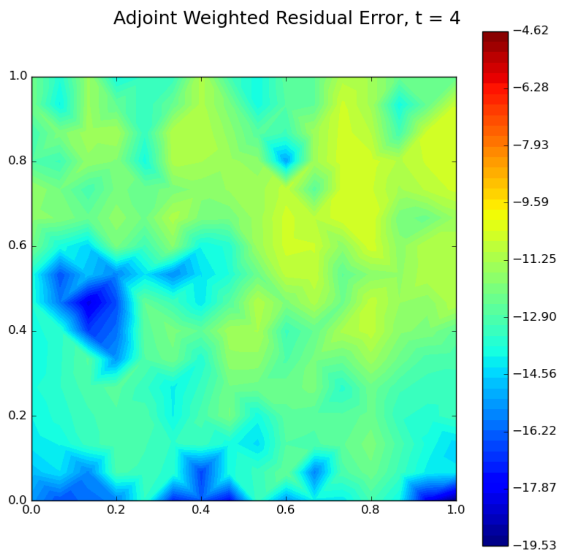

Figure 5.67: Absolute error between the discrete primal local error estimates and the POD-based primal local error estimates for $Re=500$ and $N_h=16\times16$ for the lid-driven cavity flow

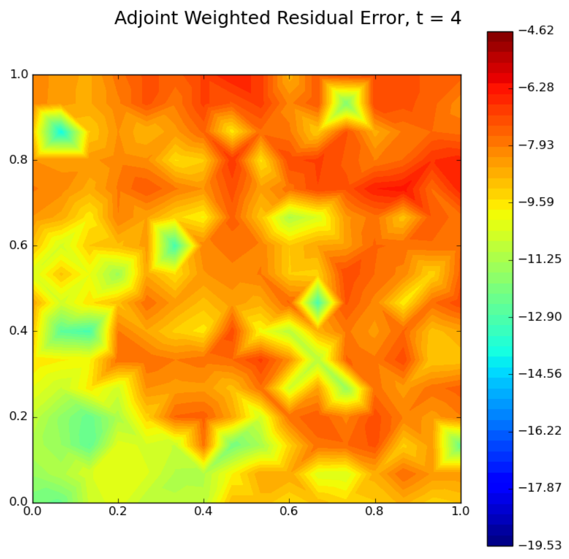

Figure 5.68: Absolute error between the discrete primal local error estimates and the autoencoder-based primal local error estimates for $Re=500$ and $N_h=16\times16$ for the lid-driven cavity flow

Equation 5.9. Next to this, the deficiency in the error has been plotted as well in Figure 5.82. The definition of the deficiency of the error estimates can be seen in Equation 5.9. From Figure 5.82, it can be seen that the error estimates for the autoencoder and the discrete primal lie close to each other for low resolutions and start showing differences when moving to higher resolutions. This is highlighted more in Figure 5.84, where the similarity of the deficiency of the reconstructed error estimates is normalized with the deficiency of the discrete primal error estimates. It should be noted that in Figure 5.84, the relative distance from a value of 1 measures how similar the values are. Therefore, when a value of 2 is seen for the resolution of 64×64, it would be as badly similar to the value 0.5 in the plot.

$$d\delta\overline{J}=\delta\overline{J}_{\text{true}}-\delta\overline{J}_{\text{est}} \quad \text{with} \quad \delta\overline{J}_{\text{true}}=\overline{J}_{\text{ref}}-\overline{J}_{H} \tag{5.9}$$

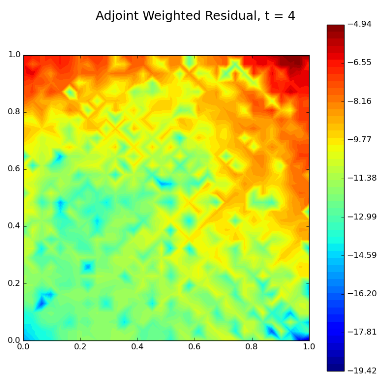

Figure 5.69:
Local Error estimate field based on the original primal for $Re = 500$ and $N_h = 32\times32$ for the lid-driven cavity flow

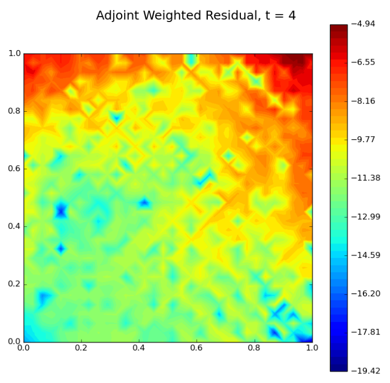

Figure 5.70:
Local Error estimate field based on the original primal for $Re = 500$ and $N_h = 32\times32$ for the lid-driven cavity flow

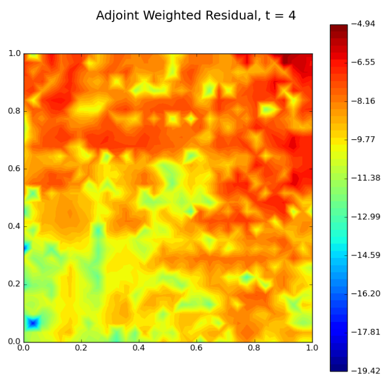

Figure 5.71:
Local Error estimate field based on the original primal for $Re = 500$ and $N_h = 32\times32$ for the lid-driven cavity flow

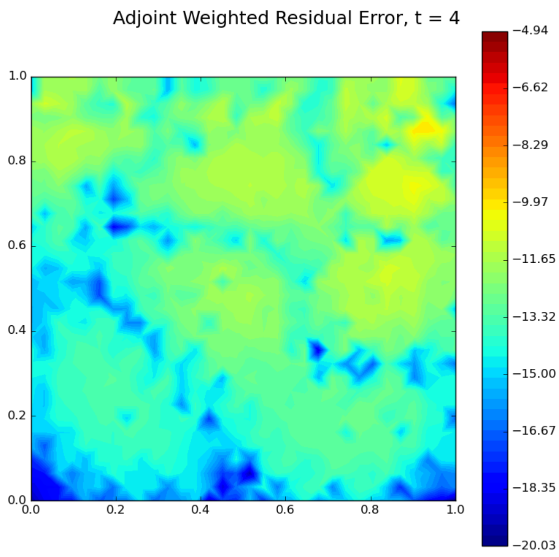

Figure 5.72: Absolute error between the discrete primal local error estimates and the POD-based primal local error estimates for $Re = 500$ and $N_h = 32\times32$ for the lid-driven cavity flow

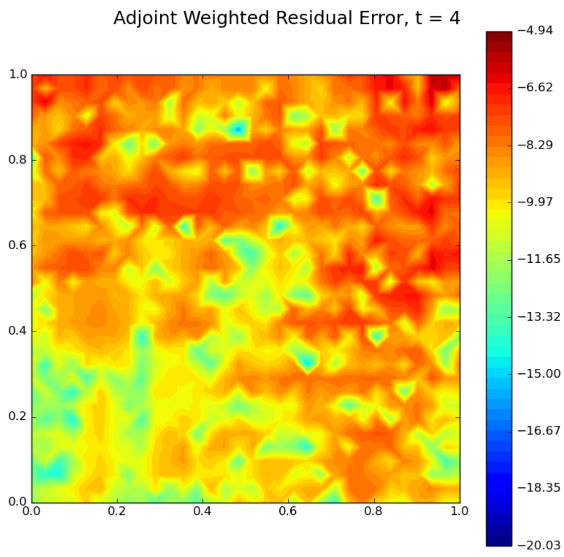

Figure 5.73: Absolute error between the discrete primal local error estimates and the autoencoder-based primal local error estimates for $Re = 500$ and $N_h = 32\times32$ for the lid-driven cavity flow

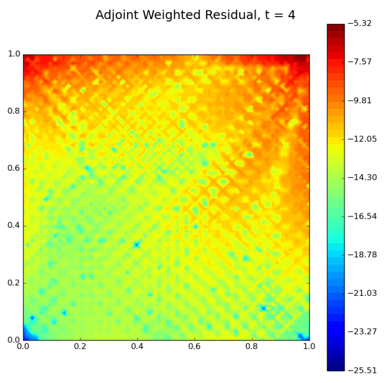

Figure 5.74:
Local Error estimate field based on the original primal for $Re = 500$ and $N_h = 64\times64$ for the lid-driven cavity flow

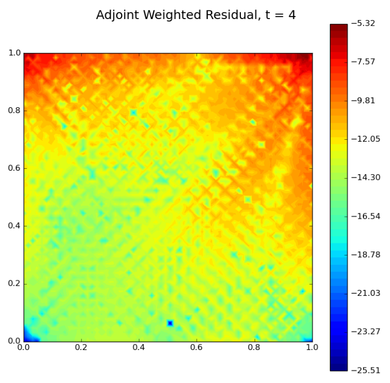

Figure 5.75:
Local Error estimate field based on the original primal for $Re = 500$ and $N_h = 64\times64$ for the lid-driven cavity flow

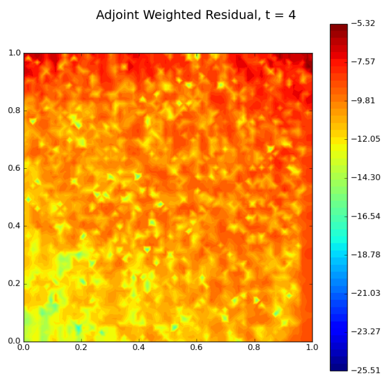

Figure 5.76:
Local Error estimate field based on the original primal for $Re = 500$ and $N_h = 64\times64$ for the lid-driven cavity flow

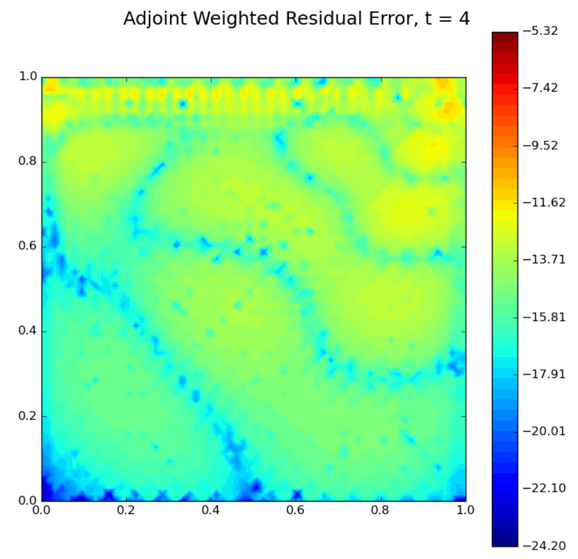

Figure 5.77: Absolute error between the discrete primal local error estimates and the POD-based primal local error estimates for $Re = 500$ and $N_h = 64 \times 64$ for the lid-driven cavity flow

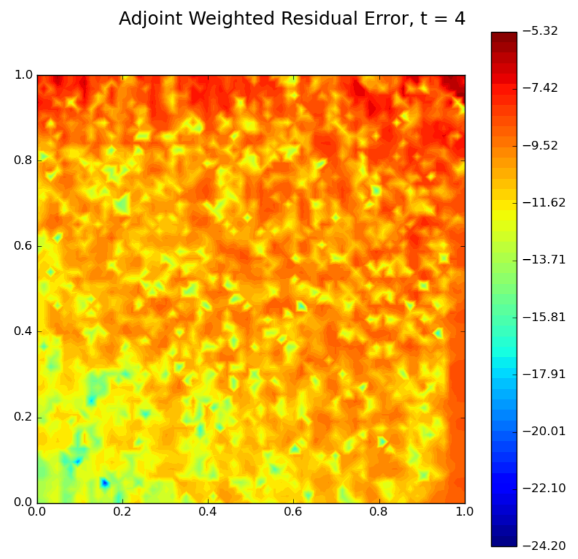

Figure 5.78: Absolute error between the discrete primal local error estimates and the autoencoder-based primal local error estimates for $Re = 500$ and $N_h = 64 \times 64$ for the lid-driven cavity flow

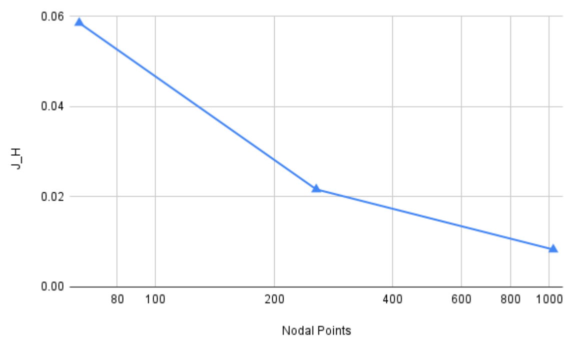

Figure 5.79: QoI value based on the coarse primal for $Re = 500$ for the lid-driven cavity flow

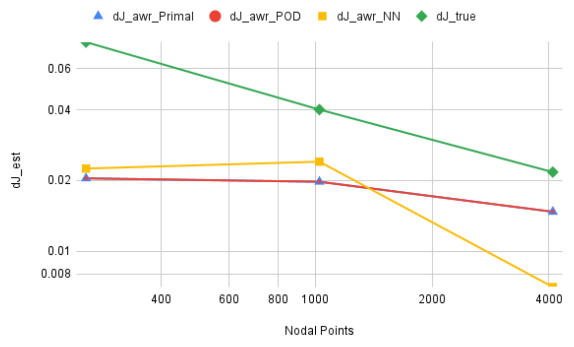

Figure 5.80: Adjoint-based error estimate for $Re = 500$ for the lid-driven cavity flow

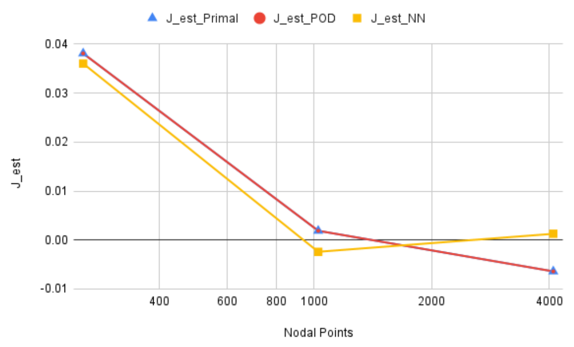

Figure 5.81: Estimated QoI on the refined space for Lid-driven cavity flow

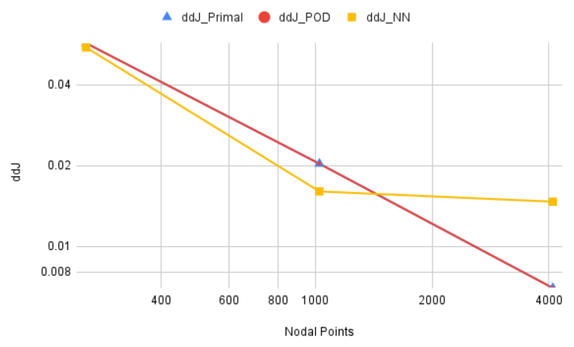

Figure 5.82: Deficiency of the adjoint-based error estimate for $Re = 500$ for the lid-driven cavity flow

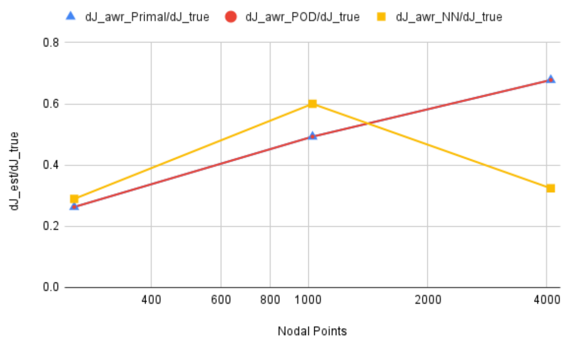

Figure 5.83:
Reconstructed Adjoint-based error estimate normalized with the true error for $Re = 500$ for the lid-driven cavity flow

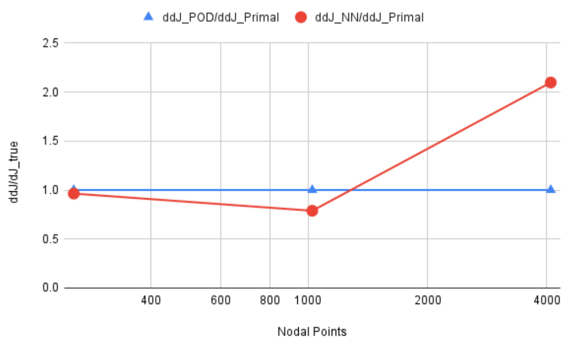

Figure 5.84: Reconstructed deficiency of the adjoint-based error estimate normalized with the deficiency of the primal error estimates for $Re = 500$ for the lid-driven cavity flow

6 Conclusion

As the final chapter of this thesis, this chapter concludes the results found and aims to provide an answer to the main research questions and secondary research objectives outlined in the report's opening chapters. The future work and recommendation section emphasizes the further actions that can be taken to improve and build upon this thesis work.

6.1. Conclusion

The thesis emphasized several restrictions on output-based mesh adaptation, particularly those relating to techniques for adjoint-based output error estimation. Two main restrictions have been accentuated, which are the high computational cost and the large storage footprint these techniques have. The focus of this thesis has been the latter point. To overcome the storage issue, a neural network autoencoder has been proposed to accurately reconstruct the primal solution. Besides this, proper orthogonal decomposition (POD) was also applied to reconstruct the primal solutions. The use of the autoencoder and POD is evaluated using the accuracy obtained for adjoint-based error estimates based on reconstructed solutions.

First, a 1D unsteady viscous Burgers problem was investigated to validate the proposed method using the method of manufactured solutions. The POD was able to reconstruct the chosen manufactured solution using only two modes. The autoencoder showed that it was able to reconstruct the primal solution to qualitatively look the same using a small latent space. However, due to the small latent space size, the reconstruction became worse as the spatial resolution increased, especially for the residual. As the adjoint-based error estimation framework uses the reconstructed primal for the adjoint calculation, and the calculation of the residual by injecting the reconstructed primal into the fine space, the error estimation was only accurate for the lower resolutions where the autoencoder was accurate for the residual as well. For the higher spatial resolutions, the reconstruction error introduced by the autoencoder starts to dominate the solution which results in more erroneous adjoint-based error estimates.

Secondly, the Navier-Stokes problem for the lid-driven cavity flow was investigated as a test case of the proposed thesis. The test case has been solved for three resolutions, namely 8, 16, and 32. The proposed method showed that reconstruction of the primal using the POD was achieved using a low number of modes due to the smoothness of the discrete primal. This resulted in a small latent space for the autoencoder. As the POD is able to capture the solution using a low number of modes, the POD has an unfair advantage when comparing the performance of the autoencoder and the POD reconstructed solutions. This resulted in the POD reconstructed primal being as accurate as the discrete primal solution when used for adjoint-based error estimation. The autoencoder for a resolution of 8 showed that it is able to estimate the adjoint-based errors but as reconstruction errors start to dominate for higher resolutions in the residual, the error estimation gets less accurate. This is because the residual is more difficult to reconstruct for higher resolution than the primal solution. The difficulties arise due to the complexity of the residual increasing as the resolution goes up, demanding a larger latent space. When the latent space is large enough to capture the residual of the primal accurately, the local error estimates based on the autoencoder reconstructed primal can be used as a first iteration error indicator for mesh refinement.

To conclude, the test cases presented in this thesis were too easy for the POD resulting in an unfair comparison between the POD and the autoencoder reconstruction methods. The framework to compare these two reconstruction methods has been established using a practical and fair compression metric which includes the model size, and an accuracy measure for the adjoint-based error estimation method. The results show that the latent space should be sufficiently large in order to gain accurate reconstruction. If the metric did not include the model size, the achieved autoencoder accuracy for the residual could be much higher.

6.2. Future work and recommendations

This thesis tested the usage of primal reconstruction using a neural network autoencoder in the context of adjoint-based error estimation. The following steps could be taken to improve and build upon this thesis work.

In this thesis, two test cases were used which were relatively easy for the POD to reconstruct. To actually compare the performance of POD reconstruction with an autoencoder reconstruction, the proposed method needs to be applied to a test case that cannot be exactly reconstructed by the POD.

Next to this, the computational capacity of the hardware used in this thesis limited the investigation. During the thesis, multiple workarounds have been used to not exceed the computational capacity. Therefore, a more powerful machine with more computational memory is highly advised for more complex cases and higher-resolution cases.

Additionally, the calculation of the fine adjoint still required the fine primal solution using the proposed method. To completely omit the need for the fine primal, a super-resolution autoencoder could be made which uses as input the coarse primal solution, compresses the solution of the primal, and uses the compressed primal to output the fine adjoint. This would completely bypass the adjoint calculation while gaining the fine adjoint cheaply.

Finally, the autoencoder in this thesis displays results that can be interpreted as nonphysical. This problem could be alleviated with the use of a physic-informed loss function instead of the Mean Squared Error used here. Note that, in order to still remain computationally cheap, this loss function cannot be expensive. This is mainly for training the neural network which evaluates this loss function often. As the function is evaluated many times, the computational cost will result in a bottleneck for training the neural network. Subsequently, the neural network training time is lower as well using a cheaply evaluated loss function.

References

[1] S. J. H. Thomas P. Hunter, “Superadjoint: Super-resolution neural networks in adjoint-based error estimation,”

[2] Y. Liu *et al.*, “A novel in situ compression method for cfd data based on generative adversarial network,” *Journal of Visualization*, vol. 22, no. 1, pp. 95–108, 2019.

[3] Y. Guo, F. Liu, Z. Cai, N. Xiao, and Z. Zhao, “Edge-based efficient search over encrypted data mobile cloud storage,” *Sensors*, vol. 18, no. 4, p. 1189, 2018.

[4] J. P. Slotnick *et al.*, “Cfd vision 2030 study: A path to revolutionary computational aerosciences,” Tech. Rep., 2014.

[5] F. Alauzet and A. Loseille, “A decade of progress on anisotropic mesh adaptation for computational fluid dynamics,” *Computer-Aided Design*, vol. 72, pp. 13–39, 2016.

[6] A. K. Runchal and M. M. Rao, “Cfd of the future: Year 2025 and beyond,” in *50 Years of CFD in Engineering Sciences*, Springer, 2020, pp. 779–795.

[7] M. Bhatia, “Adjoint-based h-adaptive calculation of generalized aerodynamic forces,” 2015. DOI: 10.2514/6.2015-0172.

[8] K. Fidkowski and D. Darmofal, “Output error estimation and adaptation in computational fluid dynamics: Overview and recent results,” in *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*, 2009, p. 1303.

[9] M. Woopen, G. May, and J. Schütz, “Adjoint-based error estimation and mesh adaptation for hybridized discontinuous galerkin methods,” *International journal for numerical methods in fluids*, vol. 76, no. 11, pp. 811–834, 2014.

[10] S. M. Kast, “An introduction to adjoints and output error estimation in computational fluid dynamics,” *arXiv preprint arXiv:1712.00693*, 2017.

[11] D. Ahlman, F. Söderlund, J. Jackson, A. Kurdila, and W. Shyy, “Proper orthogonal decomposition for time-dependent lid-driven cavity flows,” *Numerical Heat Transfer: Part B: Fundamentals*, vol. 42, no. 4, pp. 285–306, 2002.

[12] A. Olmo, A. Zamzam, A. Glaws, and R. King, “Physics-driven convolutional autoencoder approach for cfd data compressions,” *arXiv preprint arXiv:2210.09262*, 2022.

[13] K. T. Carlberg, A. Jameson, M. J. Kochenderfer, J. Morton, L. Peng, and F. D. Witherden, “Recovering missing cfd data for high-order discretizations using deep neural networks and dynamics learning,” *Journal of Computational Physics*, vol. 395, pp. 105–124, 2019.

[14] O. M. Agudelo, J. J. Espinosa, and B. De Moor, “Acceleration of nonlinear pod models: A neural network approach,” in *2009 European Control Conference (ECC)*, IEEE, 2009, pp. 1547–1552.

[15] T. Murata, K. Fukami, and K. Fukagata, “Nonlinear mode decomposition with convolutional neural networks for fluid dynamics,” *Journal of Fluid Mechanics*, vol. 882, A13, 2020.

[16] S. Kapadia, “Adjoint-based sensitivity analysis and error correction methods applied to solid oxide fuel cells,” *Journal of Fuel Cell Science and Technology*, vol. 6, p. 021 010, 2009. DOI: 10.1115/1.3005579.

[17] X. Li, S. Hulshoff, and S. Hickel, “Towards adjoint-based mesh refinement for large eddy simulation using reduced-order primal solutions: Preliminary 1d burgers study,” *Computer Methods in Applied Mechanics and Engineering*, vol. 379, p. 113 733, 2021.

[18] T. P. Miyanawala and R. K. Jaiman, “Decomposition of wake dynamics in fluid–structure interaction via low-dimensional models,” *Journal of Fluid Mechanics*, vol. 867, pp. 723–764, 2019.

[19] A. Glaws, R. King, and M. Sprague, “Deep learning for in situ data compression of large turbulent flow simulations,” *Physical Review Fluids*, vol. 5, no. 11, p. 114 602, 2020.

[20] M. Momenifar, E. Diao, V. Tarokh, and A. D. Bragg, “Dimension reduced turbulent flow data from deep vector quantisers,” *Journal of Turbulence*, vol. 23, no. 4-5, pp. 232–264, 2022.

[21] O. Hennigh, “Lat-net: Compressing lattice boltzmann flow simulations using deep neural networks,” *arXiv preprint arXiv:1705.09036*, 2017.

[22] G. Chen and K. Fidkowski, “Output-based error estimation and mesh adaptation using convolutional neural networks: Application to a scalar advection-diffusion problem,” in *AIAA Scitech 2020 Forum*, 2020, p. 1143.

[23] D. A. Venditti and D. L. Darmofal, “Grid adaptation for functional outputs: Application to two-dimensional inviscid flows,” *Journal of Computational Physics*, vol. 176, no. 1, pp. 40–69, 2002.

[24] D. A. Venditti and D. L. Darmofal, “Anisotropic grid adaptation for functional outputs: Application to two-dimensional viscous flows,” *Journal of Computational Physics*, vol. 187, no. 1, pp. 22–46, 2003.

[25] K. G. van der Zee, E. H. van Brummelen, I. Akkerman, and R. de Borst, “Goal-oriented error estimation and adaptivity for fluid–structure interaction using exact linearized adjoints,” *Computer Methods in Applied Mechanics and Engineering*, vol. 200, no. 37-40, pp. 2738–2757, 2011.

[26] J. Li, M. Zhang, J. R. Martins, and C. Shu, “Efficient aerodynamic shape optimization with deep-learning-based geometric filtering,” *AIAA Journal*, vol. 58, no. 10, pp. 4243–4259, 2020.

[27] V. Moureau, P. Domingo, and L. Vervisch, “Design of a massively parallel cfd code for complex geometries,” *Comptes Rendus Mécanique*, vol. 339, no. 2-3, pp. 141–148, 2011.

[28] G. Berkooz, P. Holmes, and J. L. Lumley, “The proper orthogonal decomposition in the analysis of turbulent flows,” *Annual review of fluid mechanics*, vol. 25, no. 1, pp. 539–575, 1993.

[29] Z. Wu, D. Laurence, S. Utyuzhnikov, and I. Afgan, “Proper orthogonal decomposition and dynamic mode decomposition of jet in channel crossflow,” *Nuclear Engineering and Design*, vol. 344, pp. 54–68, 2019.

[30] F. Sun, G. Xie, J. Song, and C. N. Markides, “Proper orthogonal decomposition and physical field reconstruction with artificial neural networks (ann) for supercritical flow problems,” *Engineering Analysis with Boundary Elements*, vol. 140, pp. 282–299, 2022.

[31] S. Li, N. Marsaglia, C. Garth, J. Woodring, J. Clyne, and H. Childs, “Data reduction techniques for simulation, visualization and data analysis,” in *Computer graphics forum*, Wiley Online Library, vol. 37, 2018, pp. 422–447.

[32] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, “Image segmentation using deep learning: A survey,” *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[33] A. T. Mohan, D. Tretiak, M. Chertkov, and D. Livescu, “Spatio-temporal deep learning models of 3d turbulence with physics informed diagnostics,” *Journal of Turbulence*, vol. 21, no. 9-10, pp. 484–524, 2020.

[34] A. Preciado Grijalva, “Generative models for the analysis of dynamical systems with applications,” 2022.

[35] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[36] O. Petrova, *Vae: Giving your autoencoder the power of imagination*, Mar. 2020. [Online]. Available: `https://www.scaleway.com/en/blog/vae-getting-more-out-of-your-autoencoder/#main`.

[37] R. Verfürth, “A posteriori error estimation and adaptive mesh-refinement techniques,” *Journal of Computational and Applied Mathematics*, vol. 50, no. 1-3, pp. 67–83, 1994.

[38] F. Dassi, S. Perotto, H. Si, and T. Streckenbach, “A priori anisotropic mesh adaptation driven by a higher dimensional embedding,” *Computer-Aided Design*, vol. 85, pp. 111–122, 2017.

[39] E. Gauci, A. Belme, A. Carabias, A. Loseille, F. Alauzet, and A. Dervieux, “A priori error-based mesh adaptation in cfd,” 2018.

[40] K. J. Fidkowski and D. L. Darmofal, “Review of output-based error estimation and mesh adaptation in computational fluid dynamics,” *AIAA journal*, vol. 49, no. 4, pp. 673–694, 2011.

[41] W. C. Tyson, K. Swirydowicz, J. M. Derlaga, C. J. Roy, and E. de Sturler, “Improved functional-based error estimation and adaptation without adjoints,” in *46th AIAA Fluid Dynamics Conference*, 2016, p. 3809.

[42] P. Benard, G. Balarac, V. Moureau, C. Dobrzynski, G. Lartigue, and Y. d’Angelo, “Mesh adaptation for large-eddy simulations in complex geometries,” *International journal for numerical methods in fluids*, vol. 81, no. 12, pp. 719–740, 2016.

[43] N. Odier *et al.*, “A mesh adaptation strategy for complex wall-modeled turbomachinery les,” *Computers & Fluids*, vol. 214, p. 104 766, 2021.

[44] R. Balasubramanian and J. C. Newman III, “Comparison of adjoint-based and feature-based grid adaptation for functional outputs,” *International journal for numerical methods in fluids*, vol. 53, no. 10, pp. 1541–1569, 2007.

[45] N. K. Yamaleev, “Minimization of the truncation error by grid adaptation,” *Journal of Computational Physics*, vol. 170, no. 2, pp. 459–497, 2001.

[46] K. T. Doetsch and K. J. Fidkowski, “Combined entropy and output-based adjoint approach for mesh refinement and error estimation,” *AIAA Journal*, vol. 57, no. 8, pp. 3213–3230, 2019.

[47] T. J. Baker, “Mesh adaptation strategies for problems in fluid dynamics,” *Finite Elements in Analysis and Design*, vol. 25, no. 3-4, pp. 243–273, 1997.

[48] J. Caldwell, P. Wanless, and A. Cook, “A finite element approach to burgers’ equation,” *Applied Mathematical Modelling*, vol. 5, no. 3, pp. 189–193, 1981.

[49] O. Botella and R. Peyret, “Benchmark spectral results on the lid-driven cavity flow,” *Computers & Fluids*, vol. 27, no. 4, pp. 421–433, 1998, ISSN: 0045-7930. DOI: `https://doi.org/10.1016/S0045-7930(98)00002-4`.

[50] T. A. AbdelMigid, K. M. Saqr, M. A. Kotb, and A. A. Aboelfarag, “Revisiting the lid-driven cavity flow problem: Review and new steady state benchmarking results using gpu accelerated code,” *Alexandria engineering journal*, vol. 56, no. 1, pp. 123–135, 2017.

[51] N. Offermans, A. Peplinski, O. Marin, and P. Schlatter, *Adjoint error estimators and adaptive mesh refinement in nek5000*, 2017.

A

Appendix A: Autoencoder Architectures

A.1. 1D Burgers Equation

n_points	lin_comp	Total variables
8	28	20018
16	14	20090
32	7	20318
48	5	21078
64	3	19790
96	2	29507
128	1	26941

Table A.1: Autoencoder variables and total parameters for the 1D burgers equation

	Type	Details	Output shape
Encoder	**Input**	Input of the data in batches	[32, 1, n_points]
	Layer 1	Conv1d(1, 32, kernel_size=3, stride=2, padding=1)	[32, 32, n_points/2]
	Activation Function 1	ReLU()	
	Layer 2	Conv1d(32, 64, kernel_size=3, stride=2, padding=1)	[32, 64, n_points/4]
	Activation Function 2	ReLU()	
	Flatten	Reshape array for Linear compression	[32, 64*n_points/4]
	Layer 3	Linear(in_features=64*n_points/4, out_features=lin_comp)	[32, lin_comp]
	Activation Function 3	ReLU()	
	Layer 4^{+}	Linear(in_features= lin_comp, out_features=1)	[32,1]
Decoder	Layer 5	Linear(in_features= 1, out_features=lin_comp)	[32, lin_comp]
	Activation Function 4	ReLU()	
	Layer 6	Linear(in_features=lin_comp, out_features=64*n_points/4)	[32, 64*n_points/4]
	Activation Function 5	ReLU()	
	Reshape	Reshape array for Convolutional reconstruction	[32, 64, n_points/4]
	Layer 7	ConvTranspose1d(64, 32, kernel_size=3, stride=2, padding=1, output_padding=1)	[32, 32, n_points/2]
	Activation Function 6	ReLU()	
	Layer 8	ConvTranspose1d(32, 1, kernel_size=3, stride=2, padding=1, output_padding=1)	[32, 1, n_points]
	Output	Output of original shape as input	

Table A.2: Autoencoder architecture for
the 1D Burgers equation. $^{+}$Note for the resolutions 48,96 and 128, out_features of Layer 4 and in_features of Layer 5 is set to 2

A.2. Lid-driven cavity flow

Reynolds Number	Resolution	Kernel_1_out	Kernel_2_out	lin_comp	r
100	8×8	13	24	22	5
	16×16	7	16	16	4
	32×32	9	14	12	5
	48×48	10	15	9	5
	64×64	10	14	9	5
	96×96	14	19	6	5
500	8×8	17	33	22	8
	16×16	14	26	17	8
	32×32	13	19	14	8
	48×48	12	20	11	8
	64×64	11	21	11	9
	96×96	4	20	11	9
900	8×8	18	36	22	9
	16×16	12	28	22	10
	32×32	12	24	14	10
	48×48	13	20	14	10
	64×64	15	19	15	11
	96×96	20	22	12	11

Table A.3: Autoencoder variables and total parameters for the lid-driven cavity flow

	Type	Details	Output
Encoder	**Input**	Input of the data in batches	[32, 3, n_points, n_points]
	Layer 1	Conv2d(1, Kernel_1_out, kernel_size=3, stride=2, padding=1)	[32, Kernel_1_out, n_points/2, n_points/2]
	Activation Function 1	ReLU()	
	Layer 2	Conv2d(Kernel_1_out, Kernel_2_out, kernel_size=3, stride=2, padding=1)	[32, Kernel_2_out, n_points/4, n_points/4]
	Activation Function 2	ReLU()	
	Flatten	Reshape array for Linear compression	[32, Kernel_2_out * n_points/4 * n_points/4]
	Layer 3	Linear(in_features=Kernel_2_out * n_points/4 * n_points/4, out_features=lin_comp)	[32, lin_comp]
	Activation Function 3	ReLU()	
	Layer 4	Linear(in_features= lin_comp, out_features=r)	[32, r]
Decoder	Layer 5	Linear(in_features= r, out_features=lin_comp)	[32, lin_comp]
	Activation Function 4	ReLU()	
	Layer 6	Linear(in_features=lin_comp, out_features=Kernel_2_out * n_points/4 * n_points/4)	[32, Kernel_2_out * n_points/4 * n_points/4]
	Activation Function 5	ReLU()	
	Reshape	Reshape array for Convolutional reconstruction	[32, Kernel_2_out, n_points/4, n_points/4]
	Layer 7	Conv2d(Kernel_2_out, Kernel_1_out, kernel_size=3, stride=2, padding=1, output_padding=1)	[32, Kernel_1_out, n_points/2, n_points/2]
	Activation Function 6	ReLU()	
	Layer 8	Conv2d(Kernel_1_out, 3, kernel_size=3, stride=2, padding=1, output_padding=1)	[32, 3, n_points, n_points]
	Output	Output of original shape as input	

Table A.4: Autoencoder architecture for the lid-driven cavity flow

B

Appendix B: 1D Burgers Solutions

B.1. 1D Burgers Primal

B.1.1. Solution

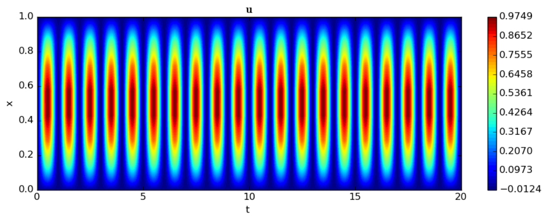

Figure B.1: 1D Burgers solution for N_H = 8

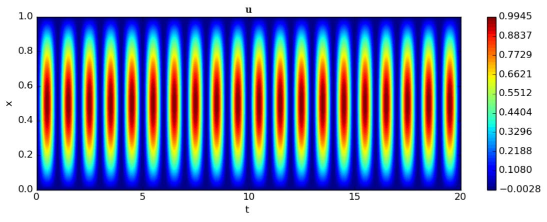

Figure B.2: 1D Burgers solution for N_H = 16

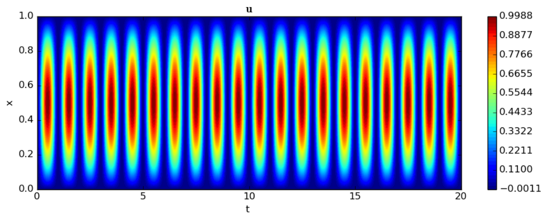

Figure B.3: 1D Burgers solution for N_H = 32

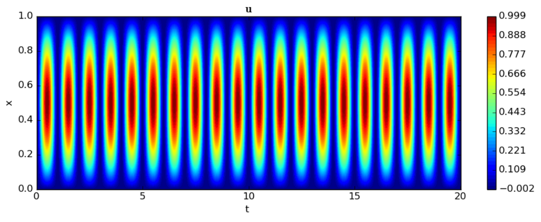

Figure B.4: 1D Burgers solution for N_H = 48

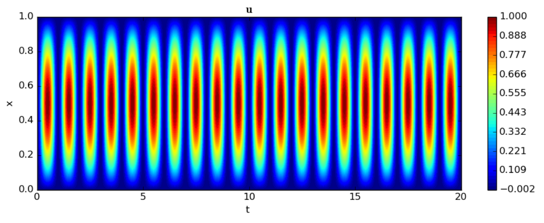

Figure B.5: 1D Burgers solution for N_H = 64

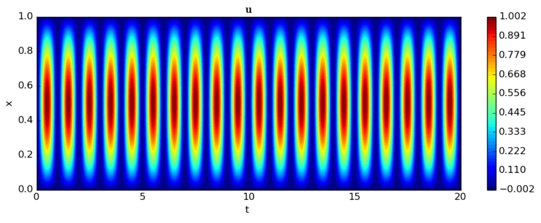

Figure B.6: 1D Burgers solution for N_H = 96

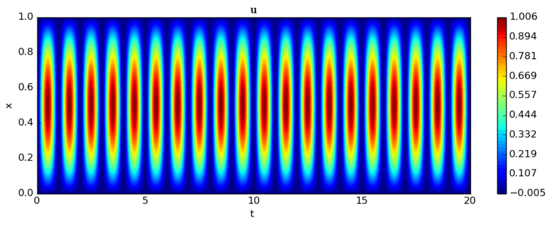

Figure B.7: 1D Burgers solution for N_H = 128

B.1.2. POD

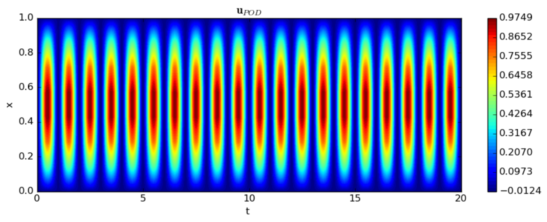

Figure B.8: 1D Burgers POD solution for N_H = 8

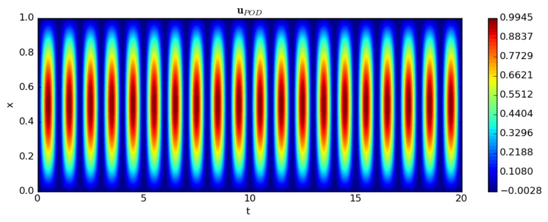

Figure B.9: 1D Burgers POD solution for N_H = 16

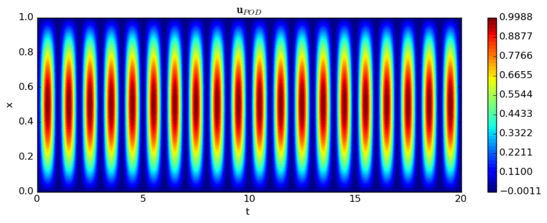

Figure B.10: 1D Burgers POD solution for N_H = 32

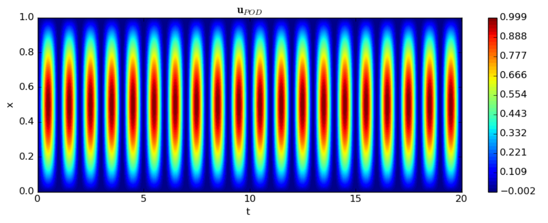

Figure B.11: 1D Burgers POD solution for N_H = 48

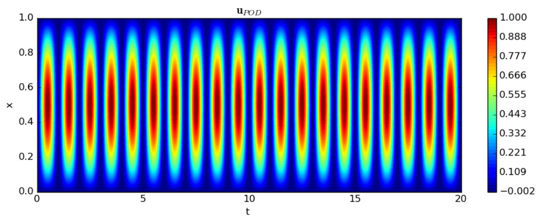

Figure B.12: 1D Burgers POD solution for N_H = 64

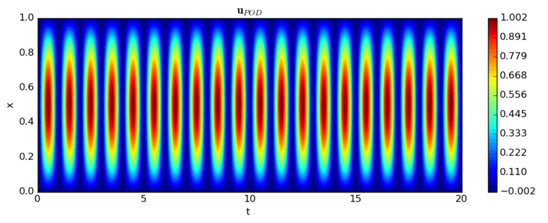

Figure B.13: 1D Burgers POD solution for N_H = 96

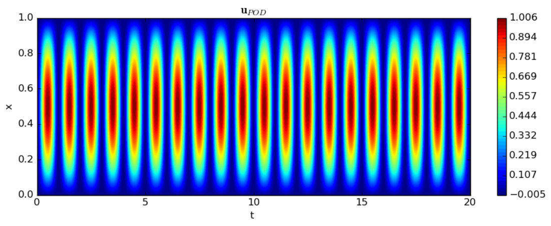

Figure B.14: 1D Burgers POD solution for N_H = 128

B.1.3. Autoencoder

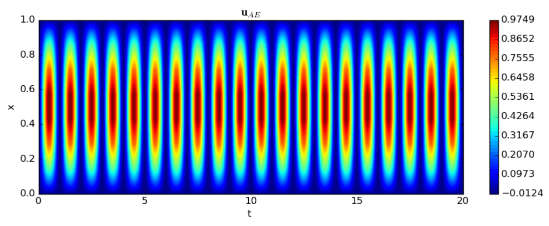

Figure B.15: 1D Burgers Autoencoder solution for N_H = 8

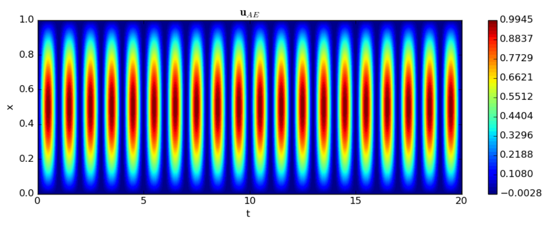

Figure B.16: 1D Burgers Autoencoder solution for N_H = 16

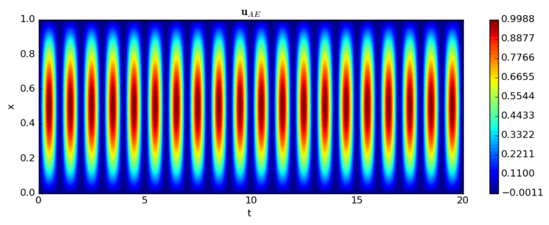

Figure B.17: 1D Burgers Autoencoder solution for N_H = 32

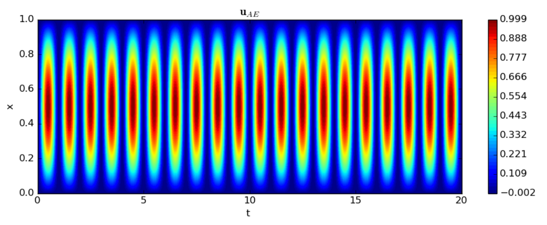

Figure B.18: 1D Burgers Autoencoder solution for N_H = 48

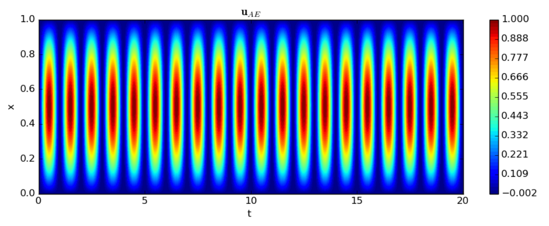

Figure B.19: 1D Burgers Autoencoder solution for N_H = 64

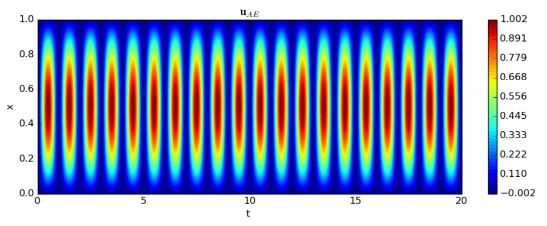

Figure B.20: 1D Burgers Autoencoder solution for N_H = 96

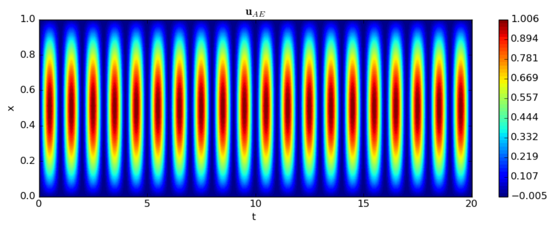

Figure B.21: 1D Burgers Autoencoder solution for N_H = 128

B.1.4. Exact Solution

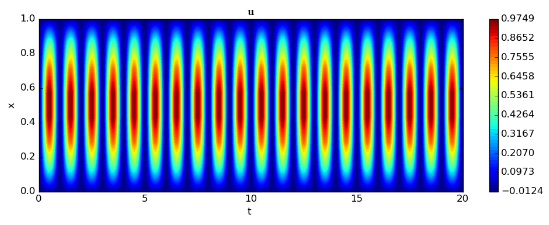

Figure B.22: 1D Burgers reference solution for N_H = 8

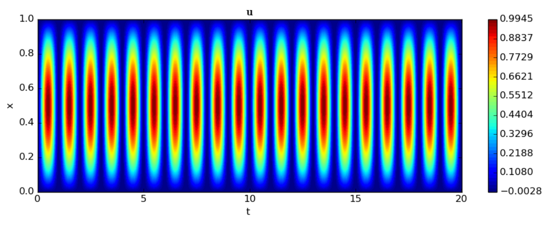

Figure B.23: 1D Burgers reference solution for N_H = 16

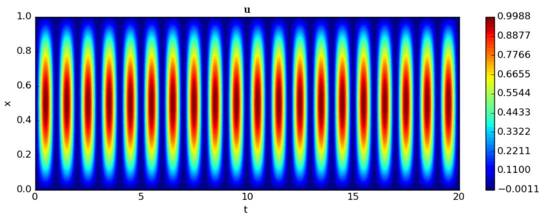

Figure B.24: 1D Burgers reference solution for N_H = 32

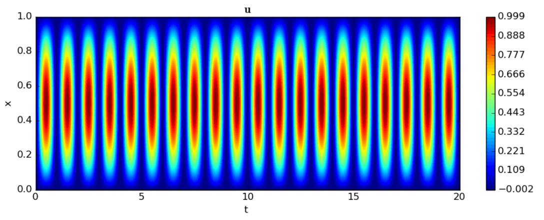

Figure B.25: 1D Burgers reference solution for N_H = 48

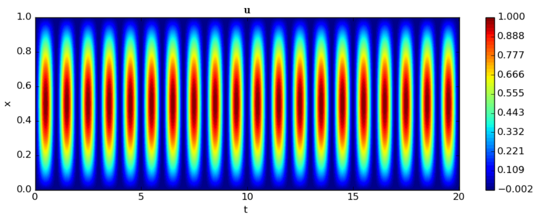

Figure B.26: 1D Burgers reference solution for N_H = 64

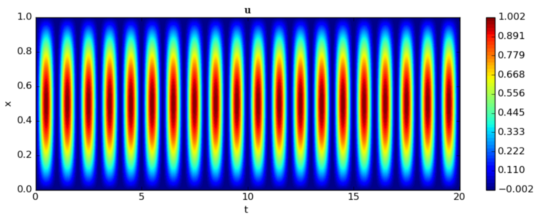

Figure B.27: 1D Burgers reference solution for N_H = 96

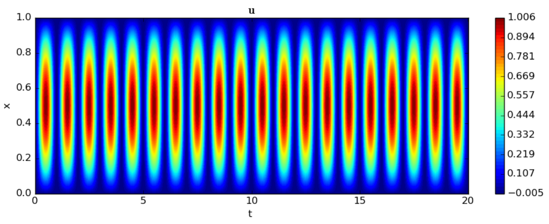

Figure B.28: 1D Burgers reference solution for N_H = 128

B.2. 1D Burgers Primal Error

B.2.1. Solution

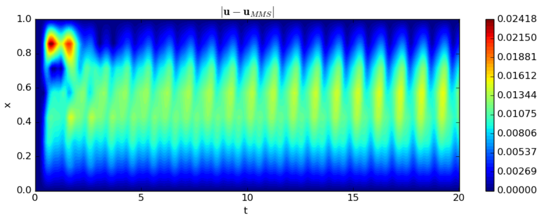

Figure B.29: 1D Burgers solution error for N_H = 8

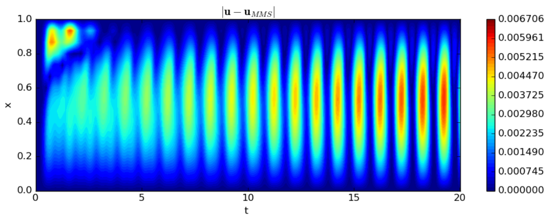

Figure B.30: 1D Burgers solution error for N_H = 16

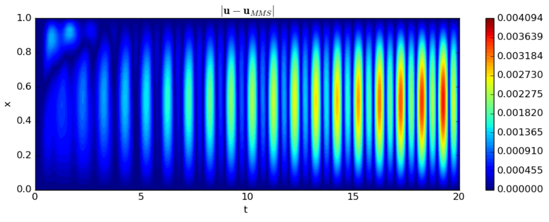

Figure B.31: 1D Burgers solution error for N_H = 32

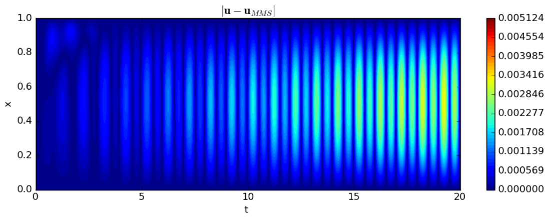

Figure B.32: 1D Burgers solution error for N_H = 48

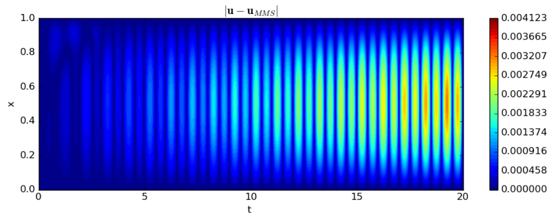
Figure B.33: 1D Burgers solution error for N_H = 64

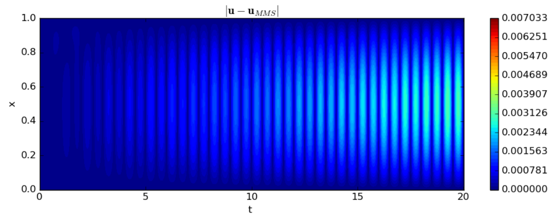
Figure B.34: 1D Burgers solution error for N_H = 96

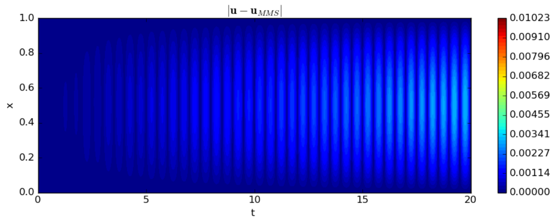
Figure B.35: 1D Burgers solution error for N_H = 128

B.2.2. POD

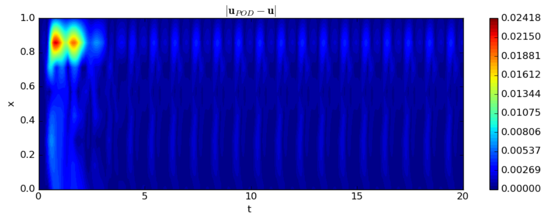
Figure B.36: 1D Burgers POD solution error for N_H = 8

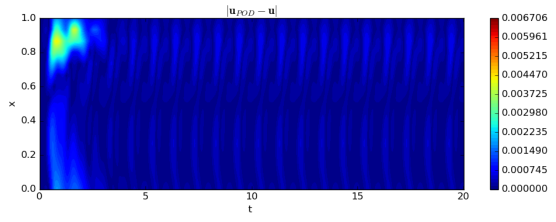
Figure B.37: 1D Burgers POD solution error for N_H = 16

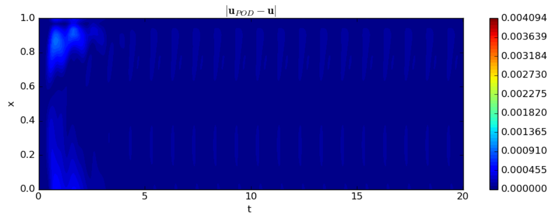
Figure B.38: 1D Burgers POD solution error for N_H = 32

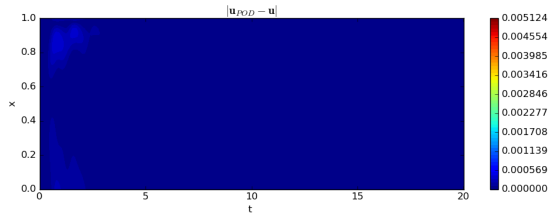
Figure B.39: 1D Burgers POD solution error for N_H = 48

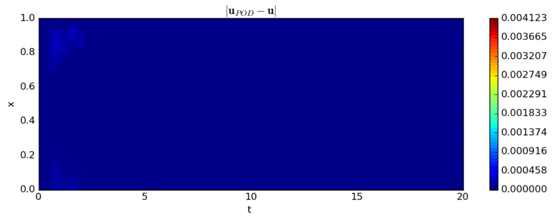
Figure B.40: 1D Burgers POD solution error for N_H = 64

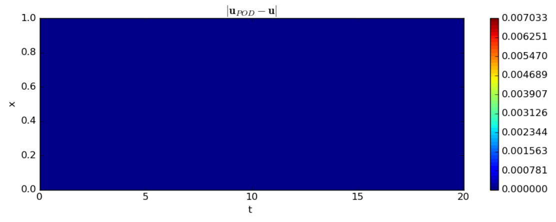
Figure B.41: 1D Burgers POD solution error for N_H = 96

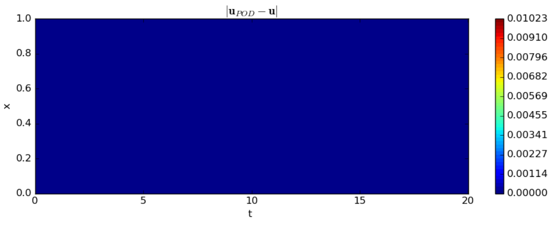

Figure B.42: 1D Burgers POD solution error for N_H = 128

B.2.3. Autoencoder

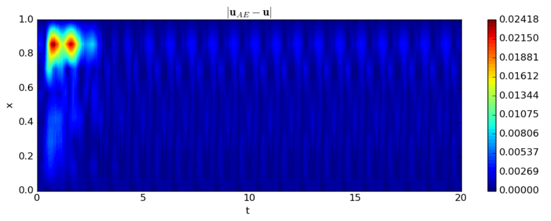

Figure B.43: 1D Burgers Autoencoder solution error for N_H = 8

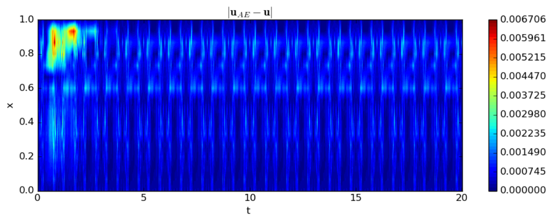

Figure B.44:
1D Burgers Autoencoder solution error for N_H = 16

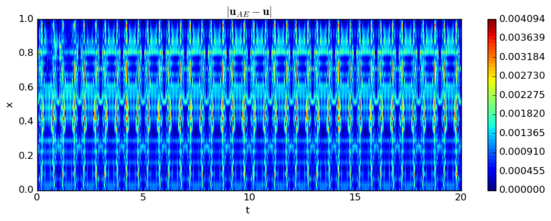

Figure B.45:
1D Burgers Autoencoder solution error for N_H = 32

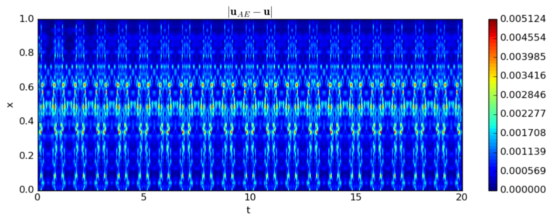

Figure B.46:
1D Burgers Autoencoder solution error for N_H = 48

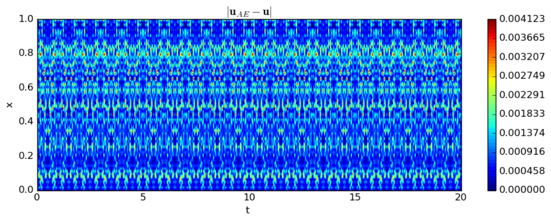

Figure B.47:
1D Burgers Autoencoder solution error for N_H = 64

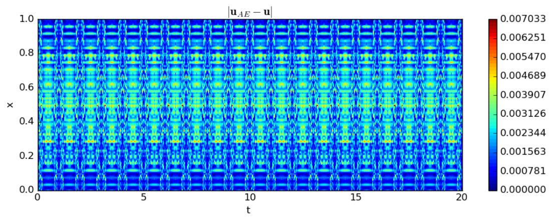

Figure B.48:
1D Burgers Autoencoder solution error for N_H = 96

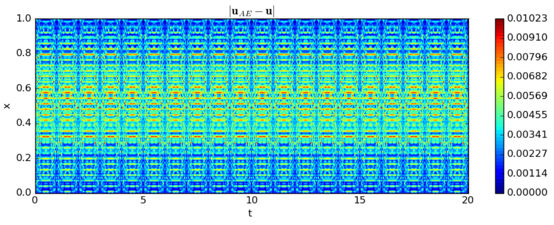

Figure B.49: 1D Burgers Autoencoder solution error for N_H = 128

B.3. 1D Burgers Adjoint Solutions

B.3.1. Primal

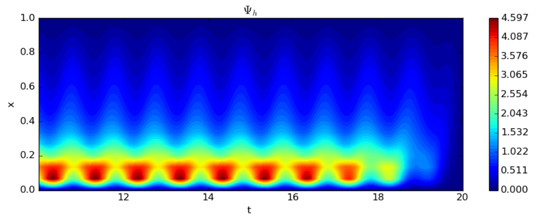

Figure B.50: 1D Burgers adjoint for N_h = 16

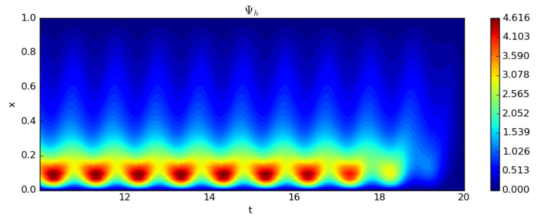

Figure B.51: 1D Burgers adjoint for N_h = 32

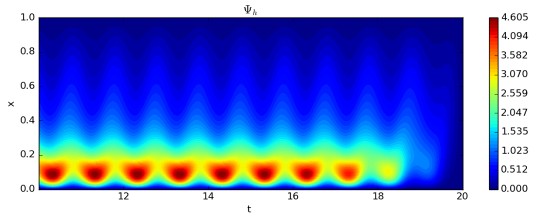

Figure B.52: 1D Burgers adjoint for N_h = 64

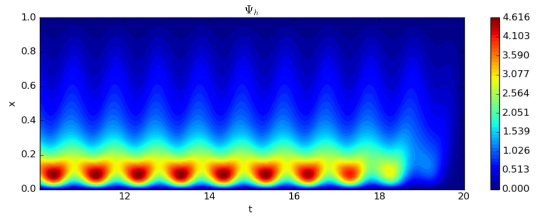

Figure B.53: 1D Burgers adjoint for N_h = 96

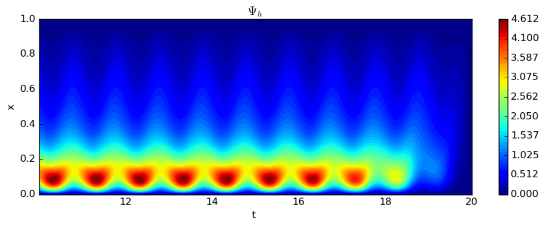

Figure B.54: 1D Burgers adjoint for N_h = 128

B.3.2. Autoencoder

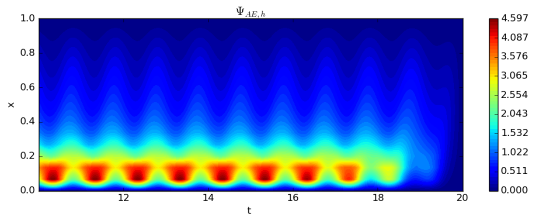

Figure B.55: 1D Burgers Autoencoder adjoint for N_h = 16

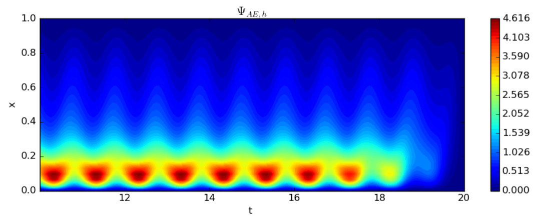

Figure B.56: 1D Burgers Autoencoder adjoint for N_h = 32

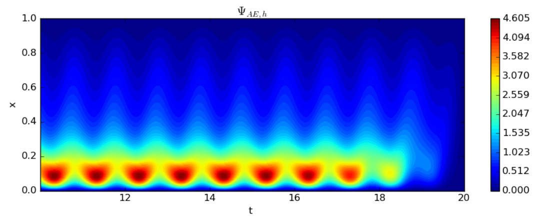
Figure B.57: 1D Burgers Autoencoder adjoint for N_h = 64

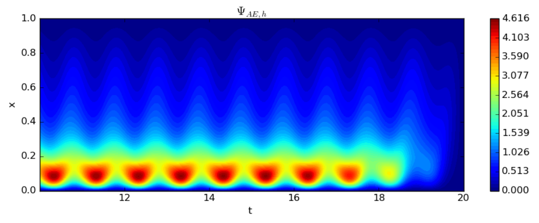
Figure B.58: 1D Burgers Autoencoder adjoint for N_h = 96

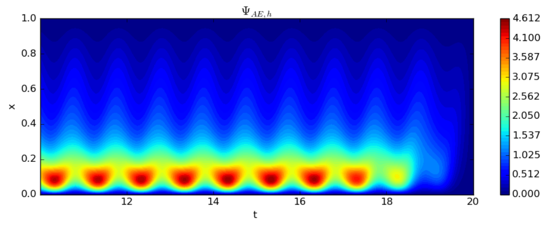
Figure B.59: 1D Burgers Autoencoder adjoint for N_h = 128

B.4. Adjoint Error

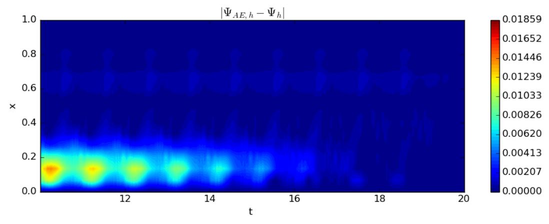
Figure B.60: 1D Burgers Autoencoder adjoint error for N_h = 16

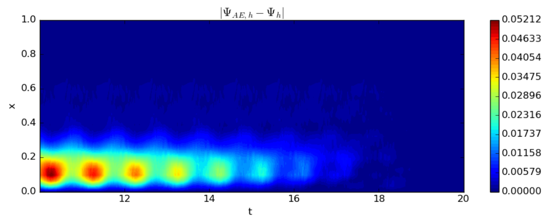
Figure B.61: 1D Burgers Autoencoder adjoint error for N_h = 32

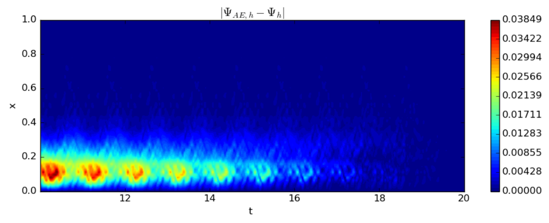
Figure B.62: 1D Burgers Autoencoder adjoint error for N_h = 64

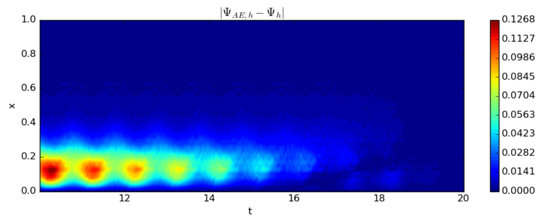
Figure B.63: 1D Burgers Autoencoder adjoint error for N_h = 96

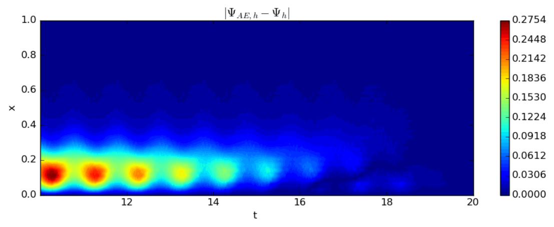
Figure B.64: 1D Burgers Autoencoder adjoint error for N_h = 128

B.5. 1D Burgers Residual

B.5.1. Primal

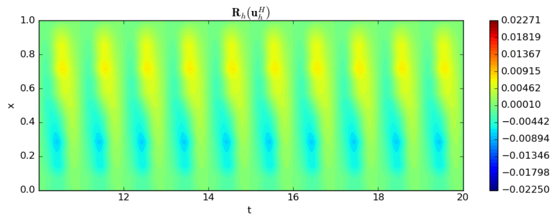

Figure B.65: 1D Burgers Residual for N_h = 16

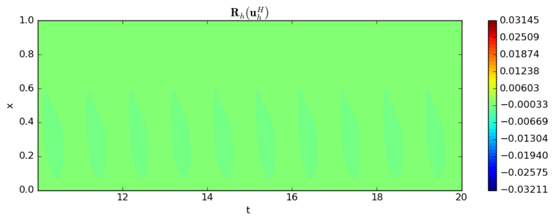

Figure B.66: 1D Burgers Residual for N_h = 32

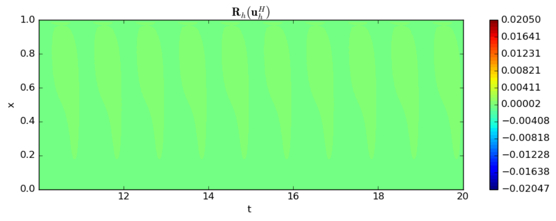

Figure B.67: 1D Burgers Residual for N_h = 64

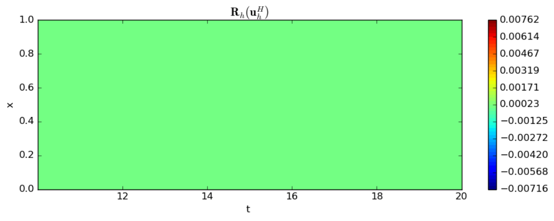

Figure B.68: 1D Burgers Residual for N_h = 96

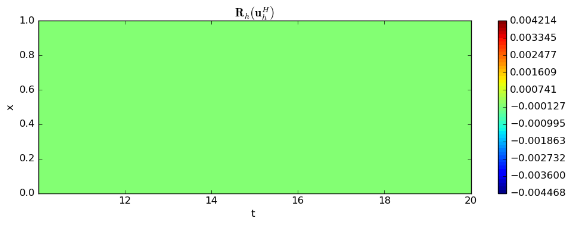

Figure B.69: 1D Burgers Residual for N_h = 128

B.5.2. Autoencoder

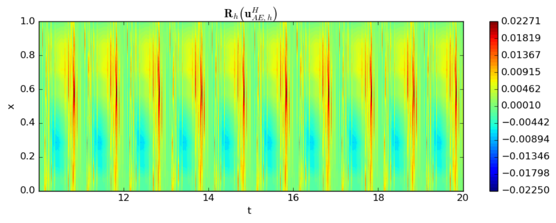

Figure B.70: 1D Burgers Autoencoder residual for N_h = 16

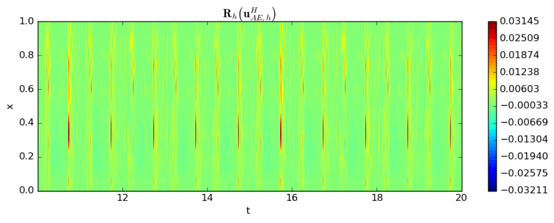

Figure B.71: 1D Burgers Autoencoder residual for N_h = 32

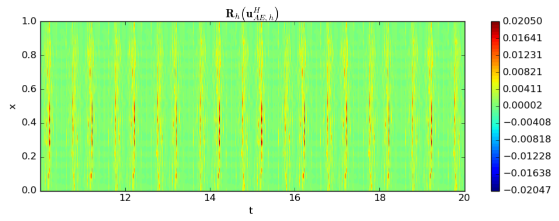

Figure B.72: 1D Burgers Autoencoder residual for N_h = 64

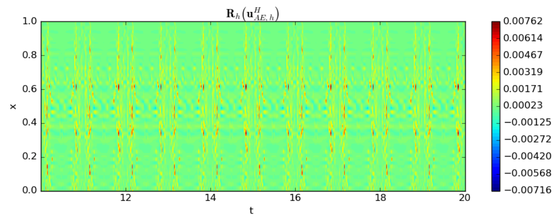

Figure B.73: 1D Burgers Autoencoder residual for N_h = 96

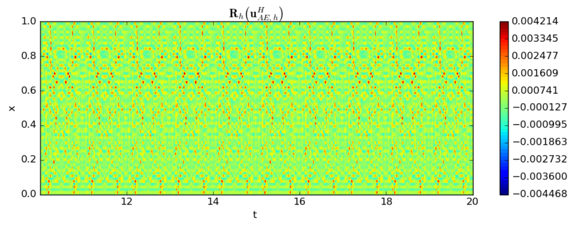

Figure B.74: 1D Burgers Autoencoder residual for N_h = 128

B.6. 1D Burgers Residual Error

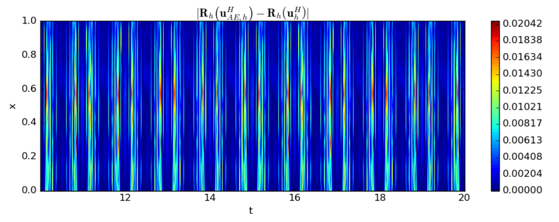

Figure B.75: 1D Burgers Autoencoder residual error for N_h = 16

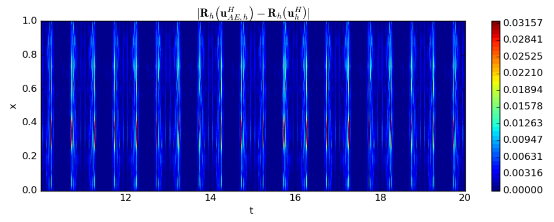

Figure B.76: 1D Burgers Autoencoder residual error for N_h = 32

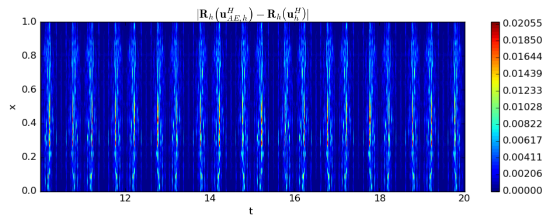

Figure B.77: 1D Burgers Autoencoder residual error for N_h = 64

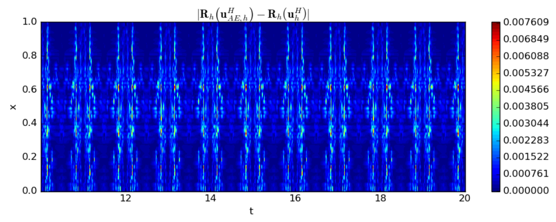

Figure B.78: 1D Burgers Autoencoder residual error for N_h = 96

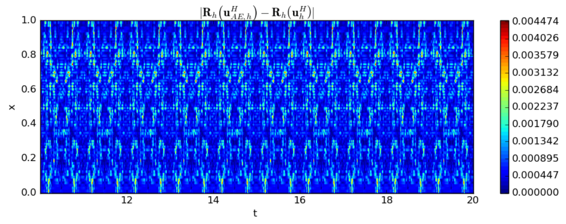

Figure B.79: 1D Burgers Autoencoder residual error for N_h = 128

B.7. 1D Burgers Adjoint weighted residual fields

B.7.1. Primal

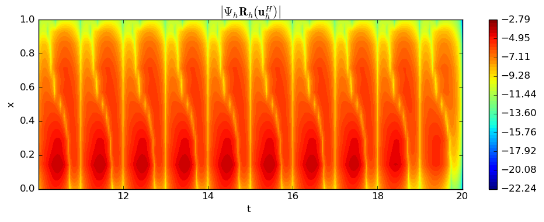

Figure B.80:
1D Burgers adjoint weighted residual field for N_h = 16

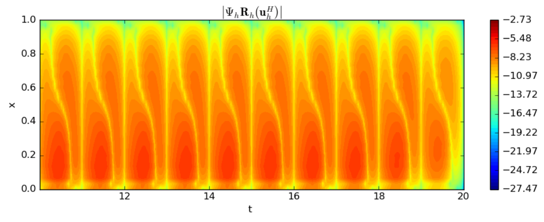

Figure B.81:
1D Burgers adjoint weighted residual field for N_h = 32

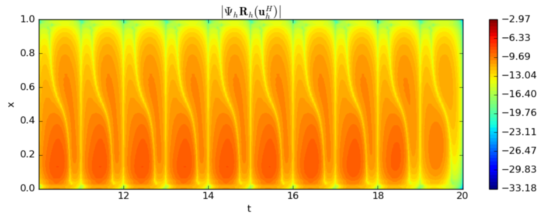

Figure B.82:
1D Burgers adjoint weighted residual field for N_h = 64

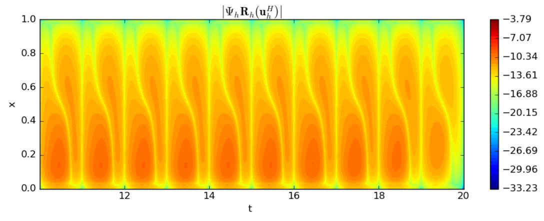

Figure B.83:
1D Burgers adjoint weighted residual field for N_h = 96

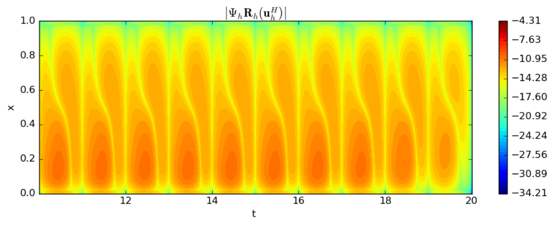

Figure B.84: 1D Burgers adjoint weighted residual field for N_h = 128

B.7.2. Autoencoder

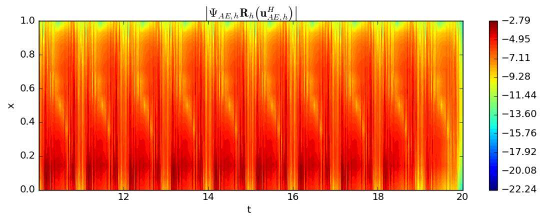

Figure B.85: 1D Burgers
Autoencoder adjoint weighted residual field for N_h = 16

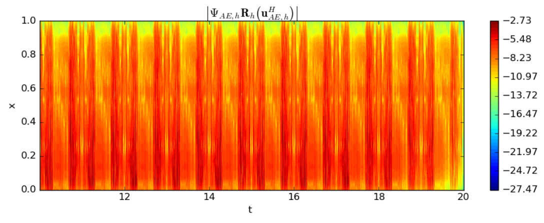

Figure B.86: 1D Burgers
Autoencoder adjoint weighted residual field for N_h = 32

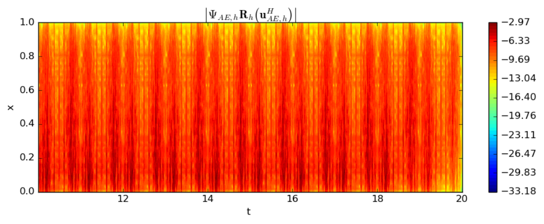
Figure B.87: 1D Burgers
Autoencoder adjoint weighted residual field for N_h = 64

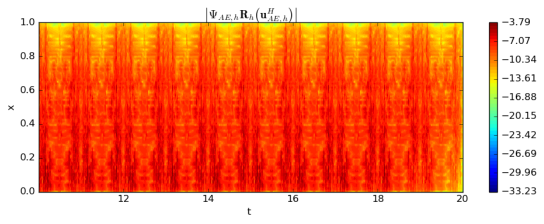
Figure B.88: 1D Burgers
Autoencoder adjoint weighted residual field for N_h = 96

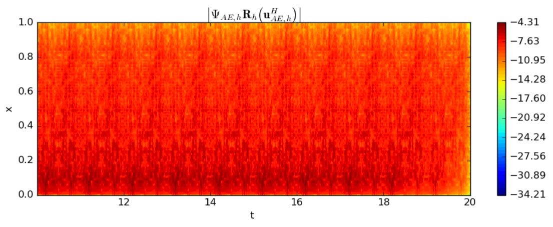
Figure B.89: 1D Burgers Autoencoder adjoint weighted residual field for N_h = 128

B.8. 1D Burgers Adjoint weighted residual error

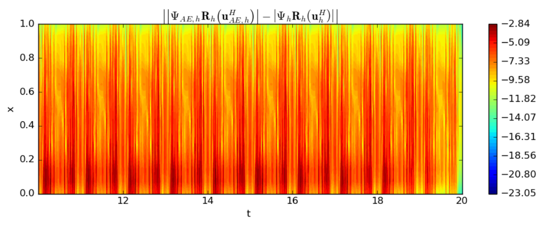
Figure B.90: 1D Burgers
Autoencoder adjoint weighted residual error for N_h = 16

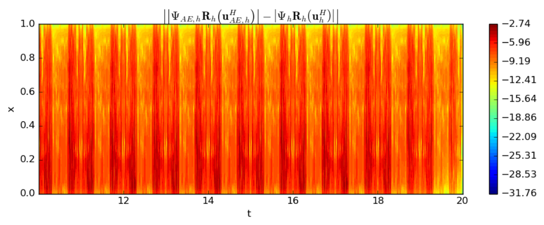
Figure B.91: 1D Burgers
Autoencoder adjoint weighted residual error for N_h = 32

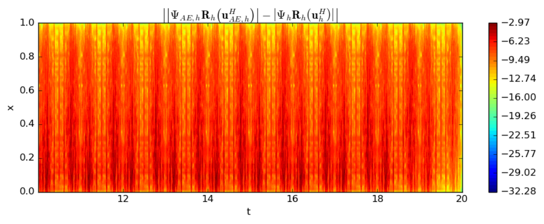
Figure B.92: 1D Burgers
Autoencoder adjoint weighted residual error for N_h = 64

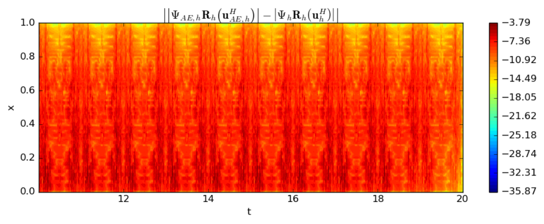
Figure B.93: 1D Burgers
Autoencoder adjoint weighted residual error for N_h = 96

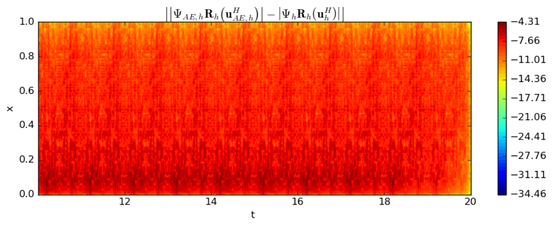

Figure B.94: 1D Burgers Autoencoder adjoint weighted residual error for N_h = 128

B.9. Train and test losses for 1D Burgers Autoencoders

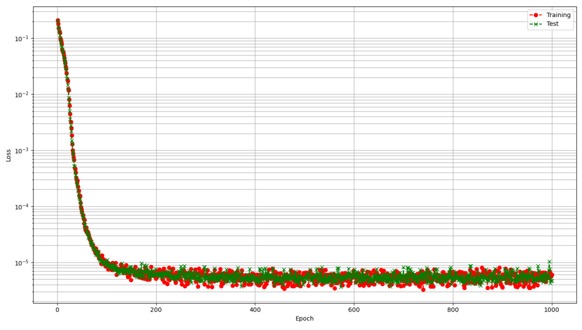

Figure B.95: 1D Burgers train and test losses for N_H = 8

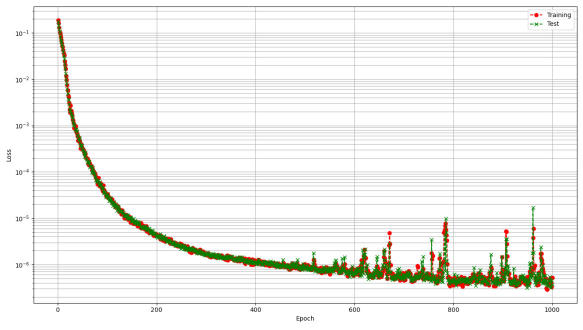

Figure B.96: 1D Burgers train and test losses for N_H = 16

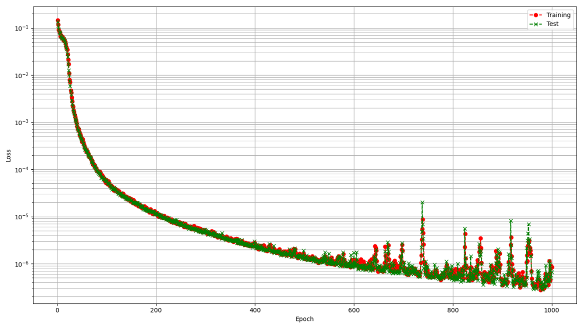

Figure B.97: 1D Burgers train and test losses for N_H = 32

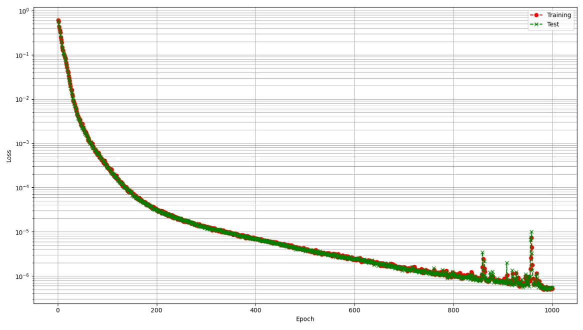

Figure B.98: 1D Burgers train and test losses for N_H = 48

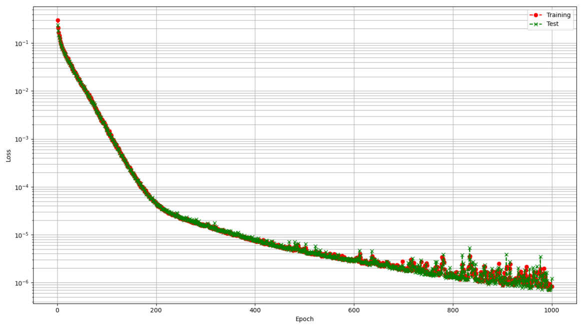

Figure B.99: 1D Burgers train and test losses for N_H = 64

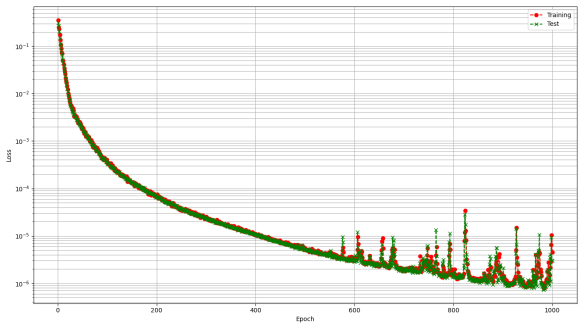

Figure B.100: 1D Burgers train and test losses for N_H = 96

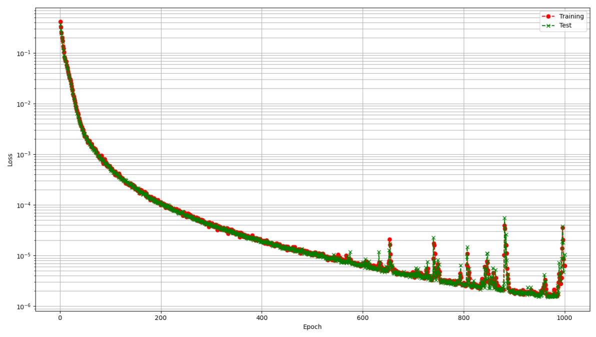

Figure B.101: 1D Burgers train and test losses for N_H = 96

C

Appendix C: Lid-driven Cavity Flow Solutions

C.1. Lid-driven Cavity Flow Primal

C.1.1. Solution

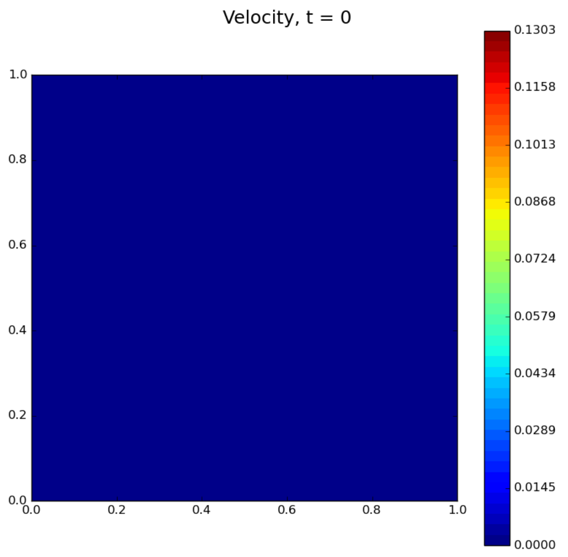

Figure C.1: Velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 0

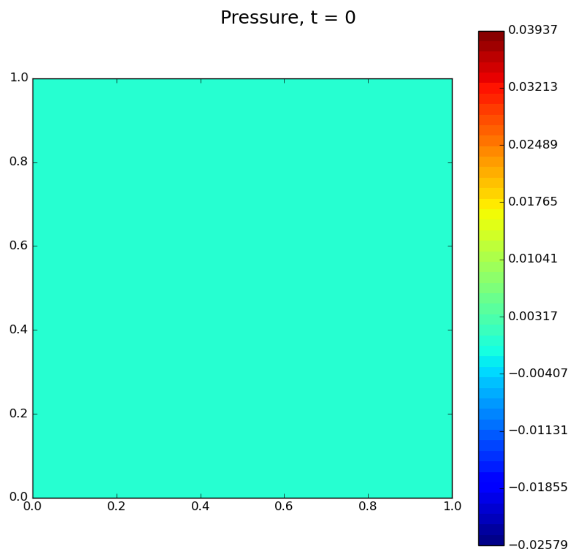

Figure C.2: Pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 0

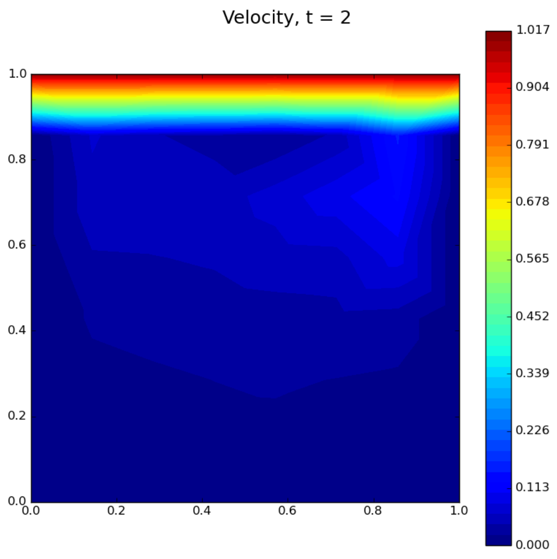

Figure C.3: Velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 2

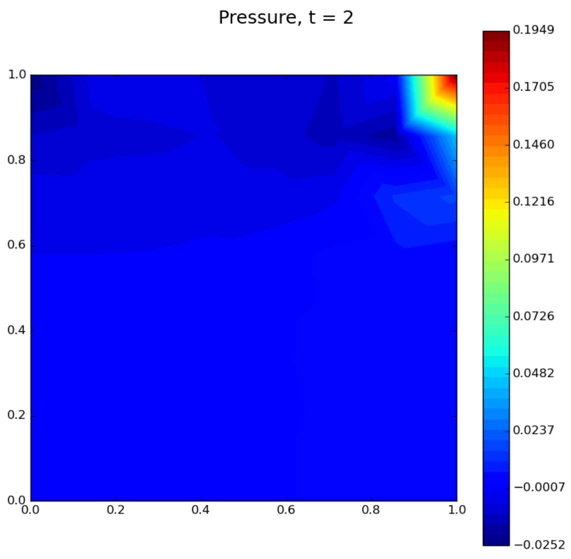

Figure C.4: Pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 2

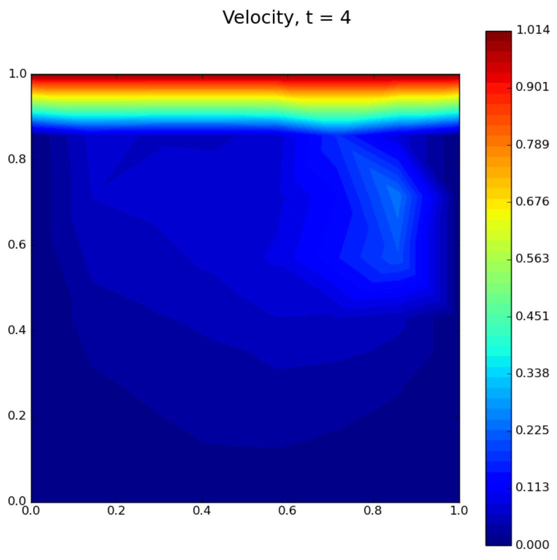

Figure C.5: Velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 4

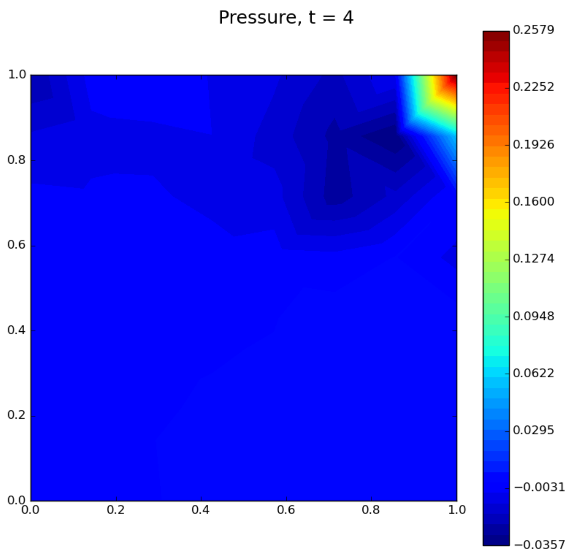

Figure C.6: Pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 4

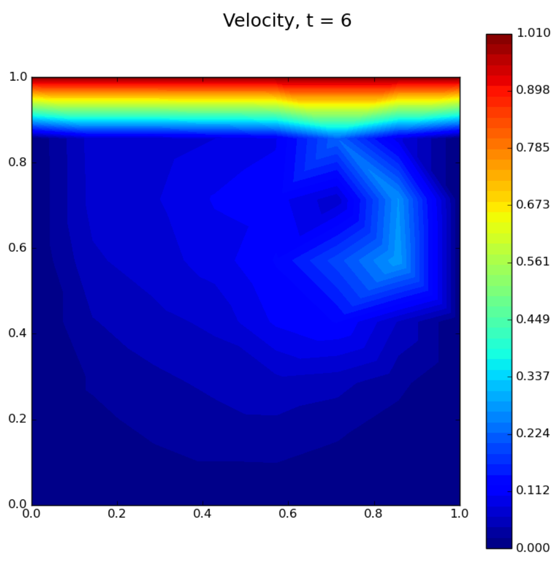

Figure C.7: Velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 6

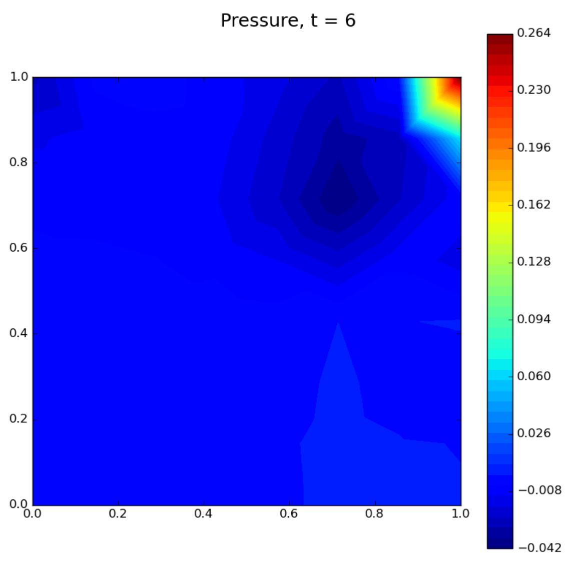

Figure C.8: Pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 6

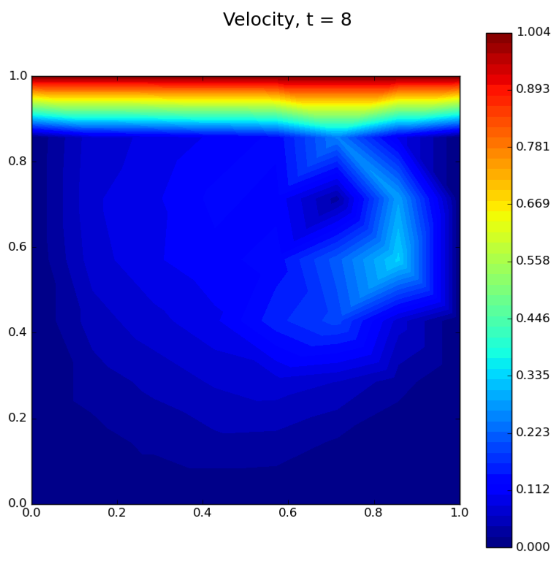

Figure C.9: Velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 8

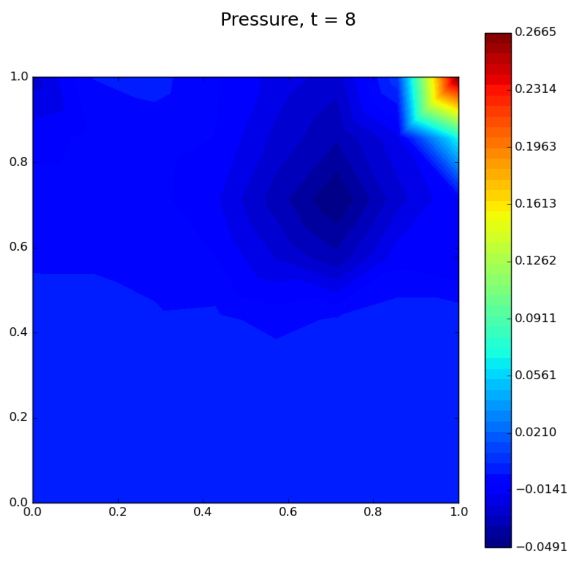

Figure C.10: Pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 8

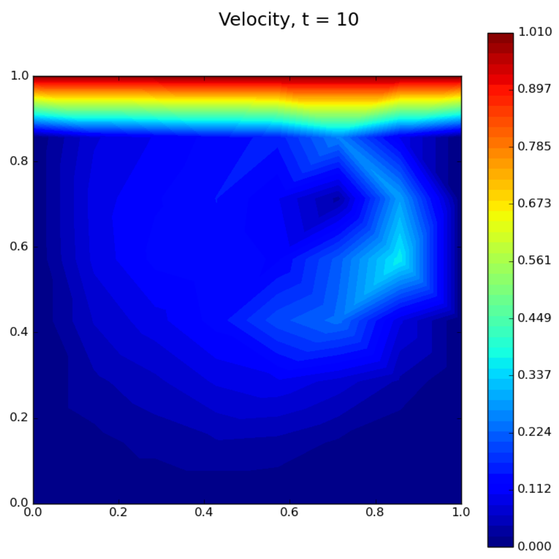

Figure C.11: Velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 10

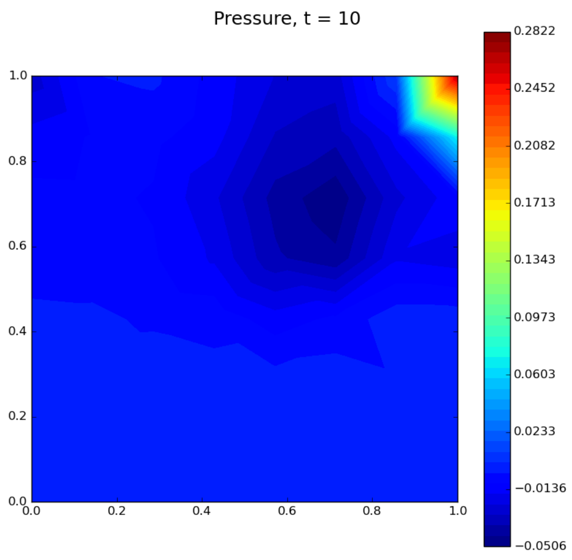

Figure C.12:
Pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 10

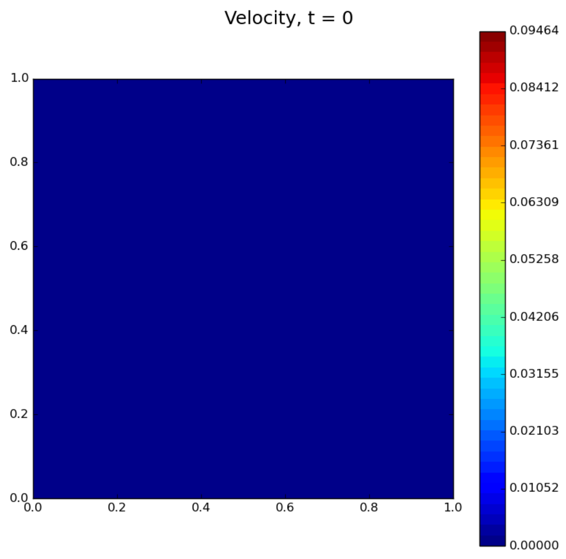

Figure C.13: Velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 0

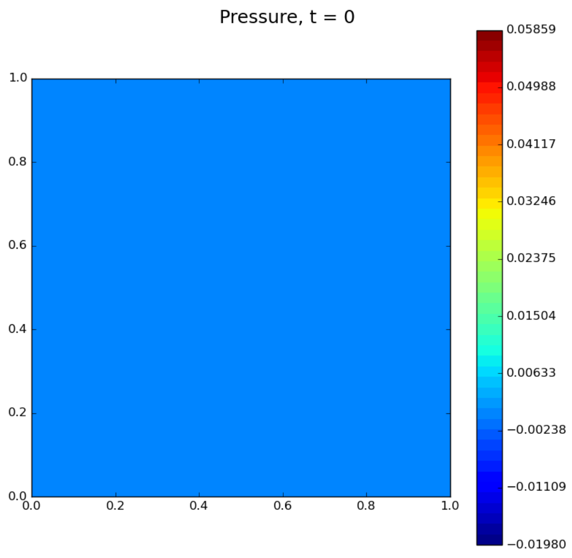

Figure C.14:
Pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 0

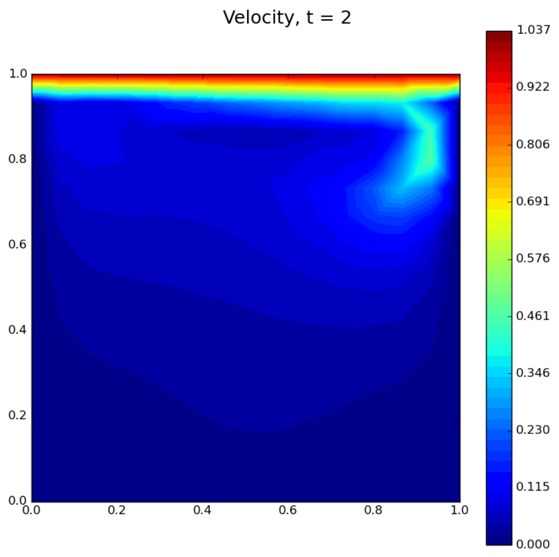

Figure C.15: Velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 2

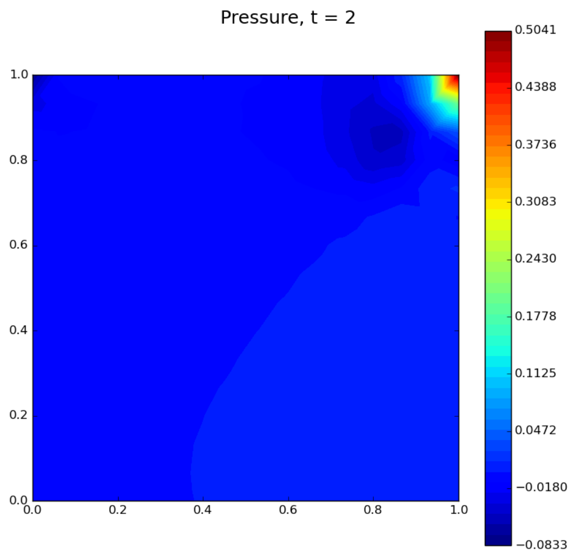

Figure C.16:
Pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 2

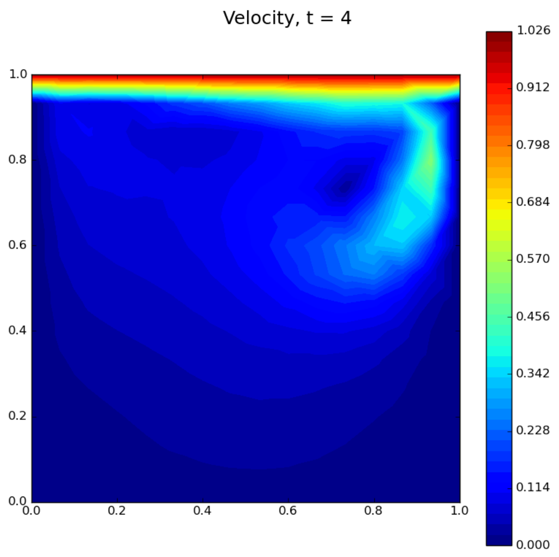

Figure C.17: Velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 4

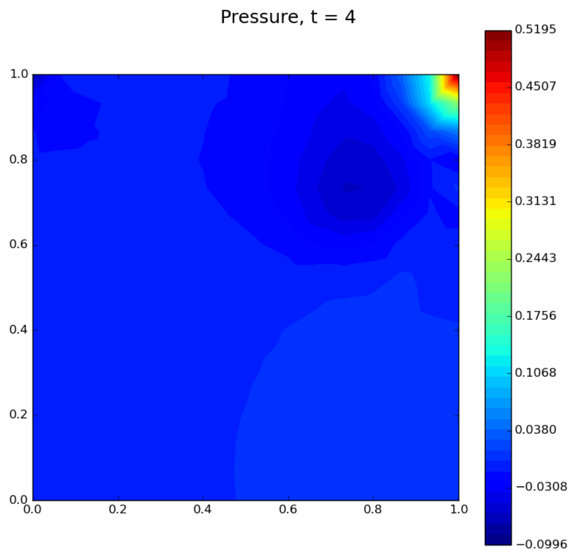

Figure C.18:
Pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 4

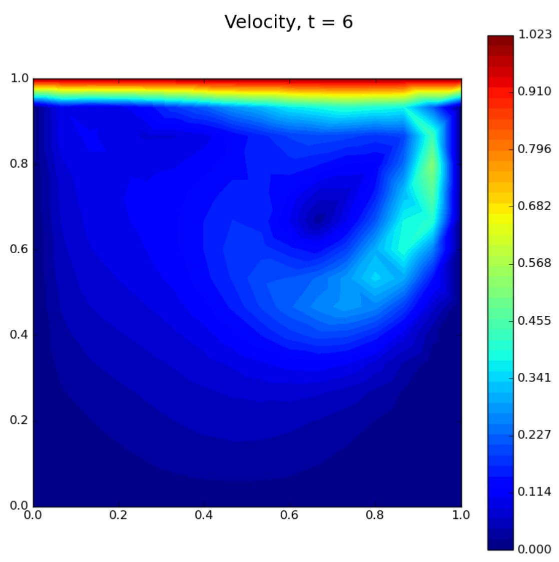

Figure C.19: Velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 6

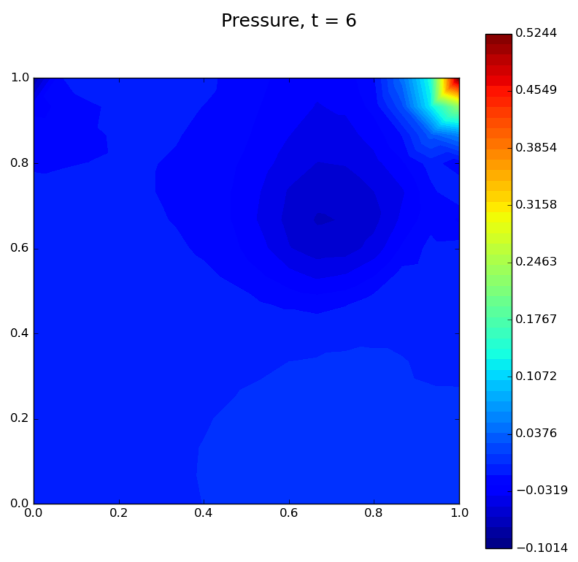

Figure C.20:
Pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 6

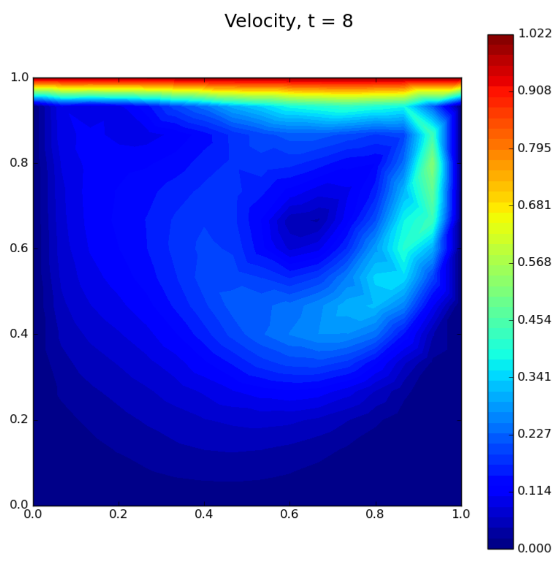

Figure C.21: Velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 8

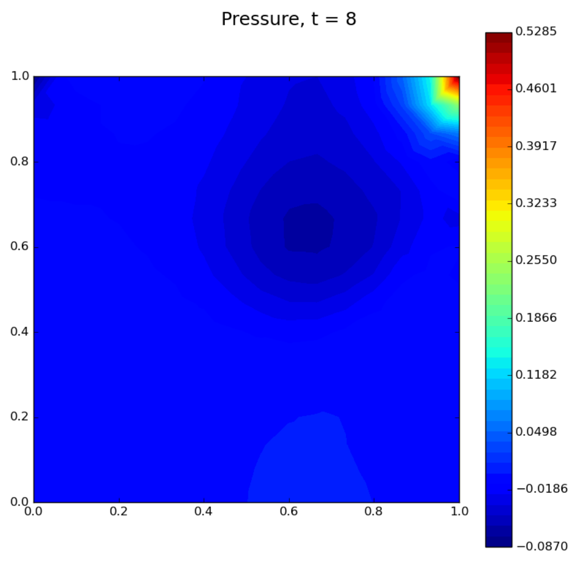

Figure C.22:
Pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 8

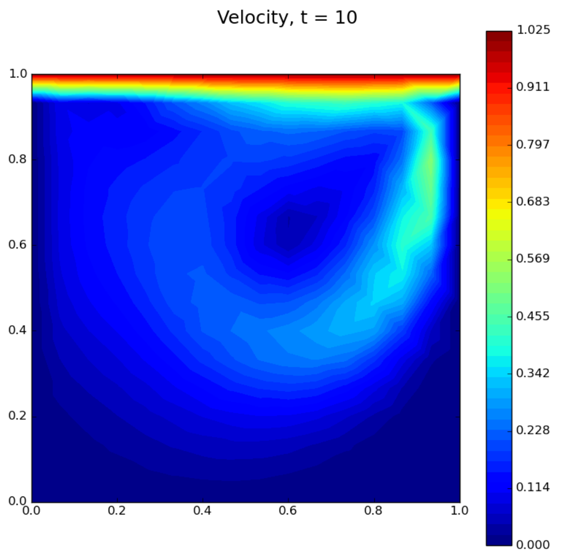

Figure C.23:
Velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 10

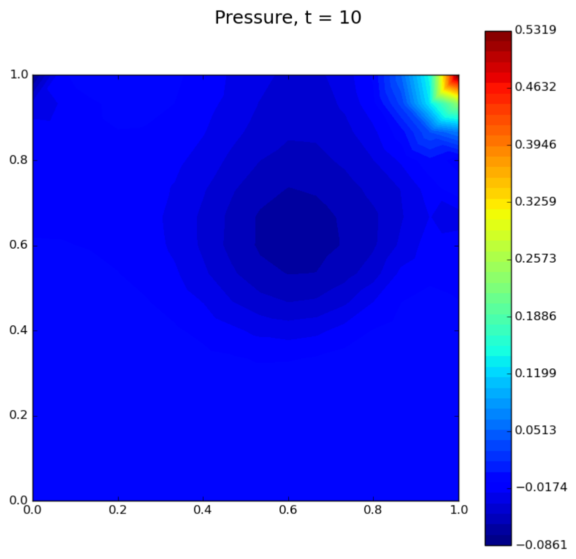

Figure C.24:
Pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 10

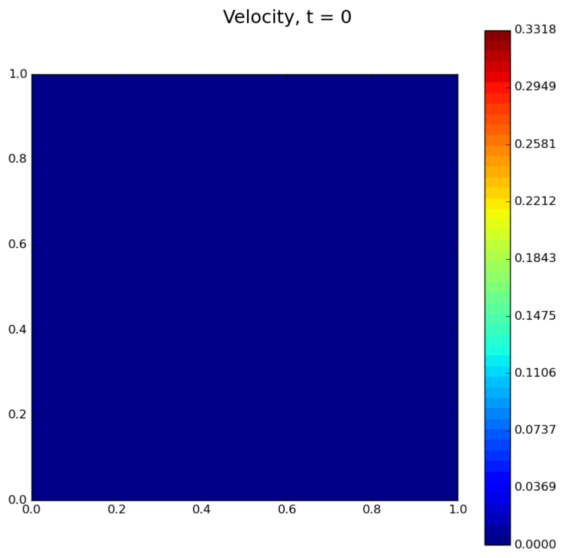

Figure C.25: Velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 0

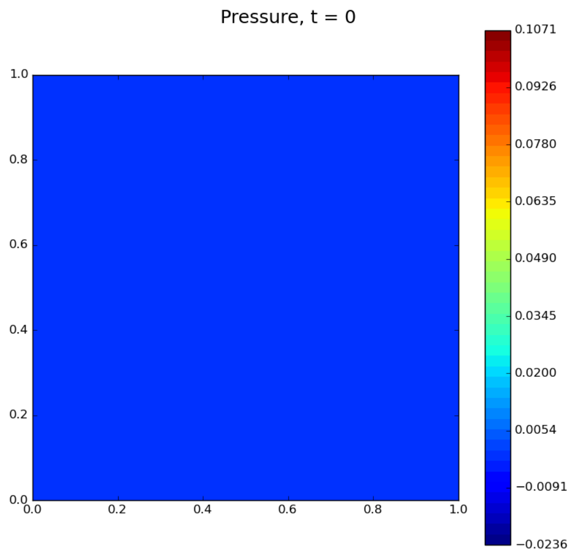

Figure C.26:
Pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 0

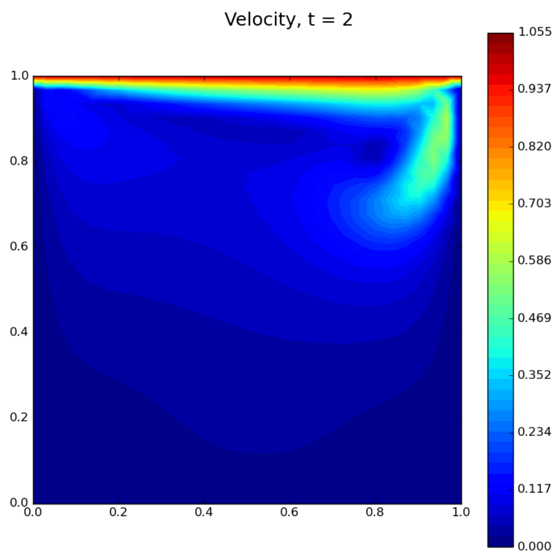

Figure C.27: Velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 2

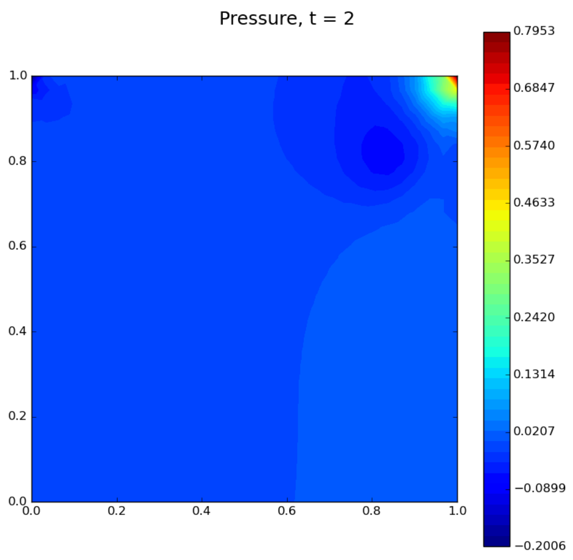

Figure C.28:
Pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 2

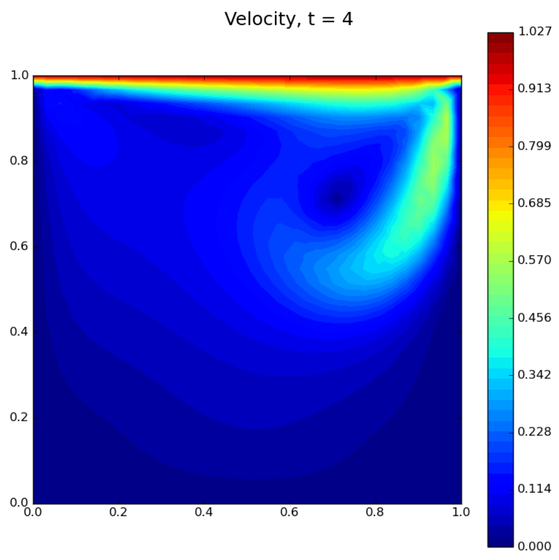

Figure C.29: Velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 4

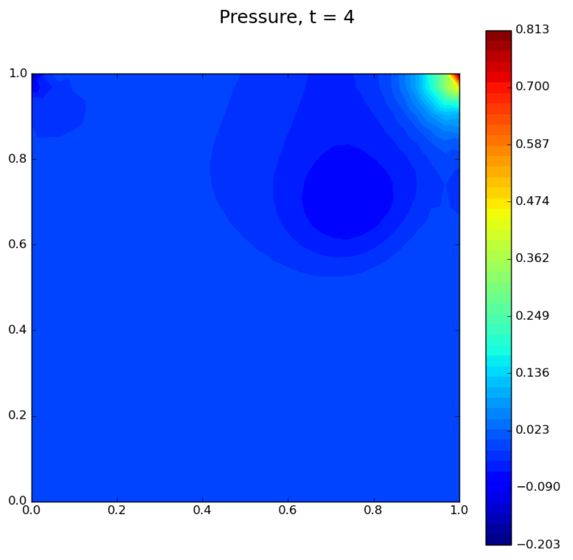

Figure C.30:
Pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 4

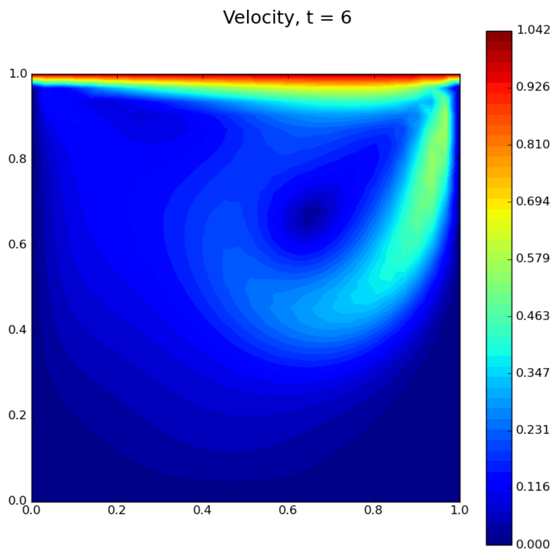

Figure C.31: Velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 6

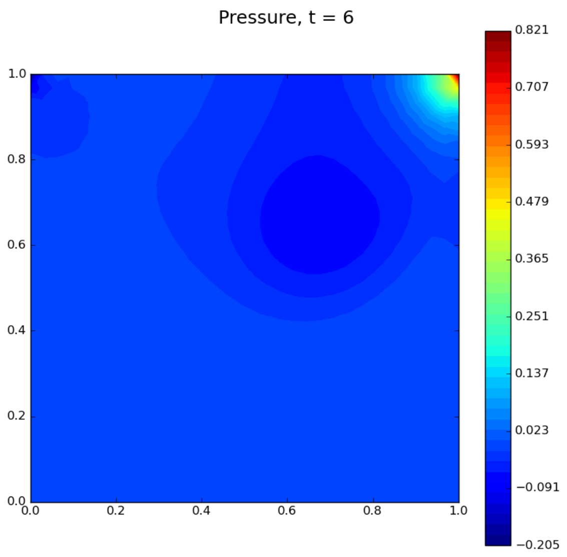

Figure C.32:
Pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 6

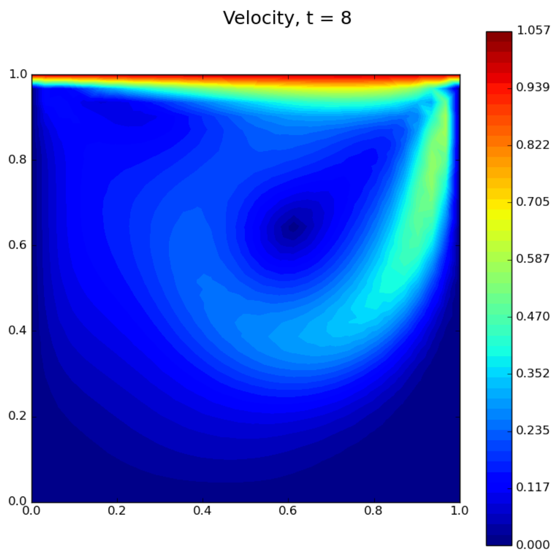

Figure C.33: Velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 8

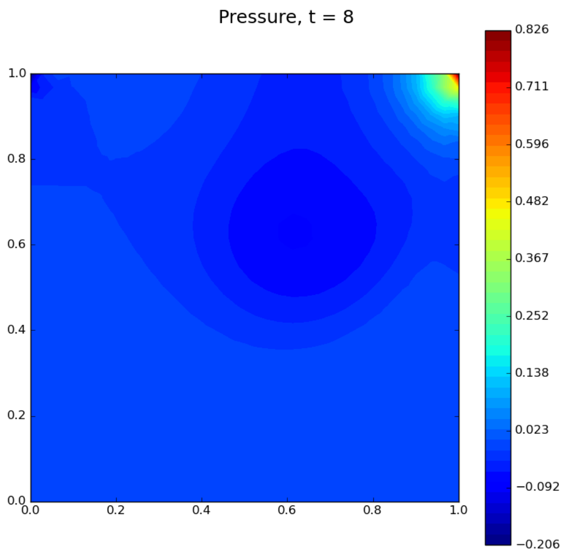

Figure C.34:
Pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 8

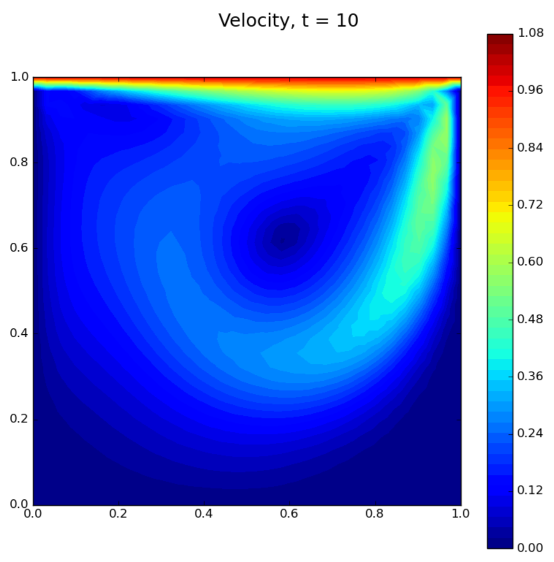

Figure C.35:
Velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 10

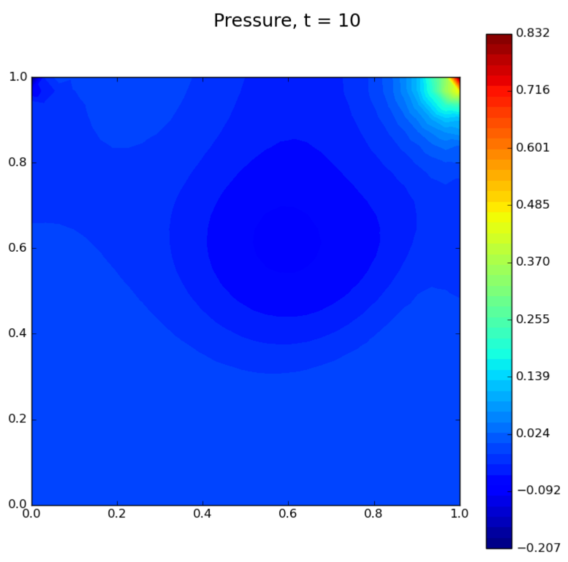

Figure C.36:
Pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 10

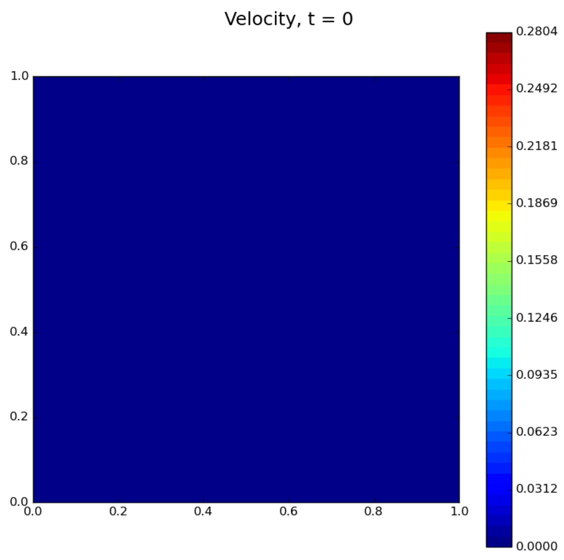

Figure C.37: Velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 0

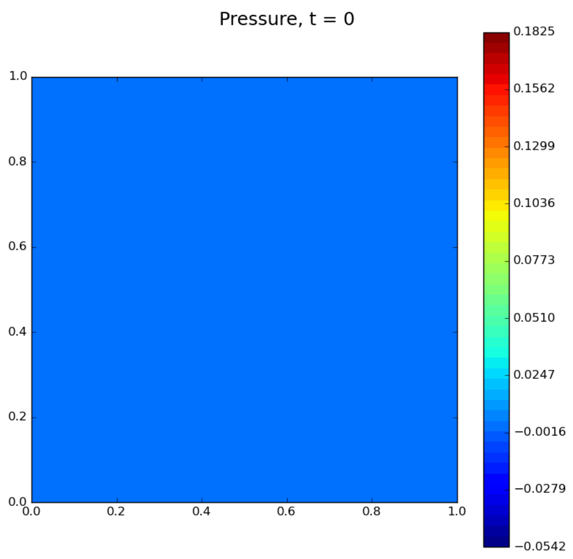

Figure C.38:
Pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 0

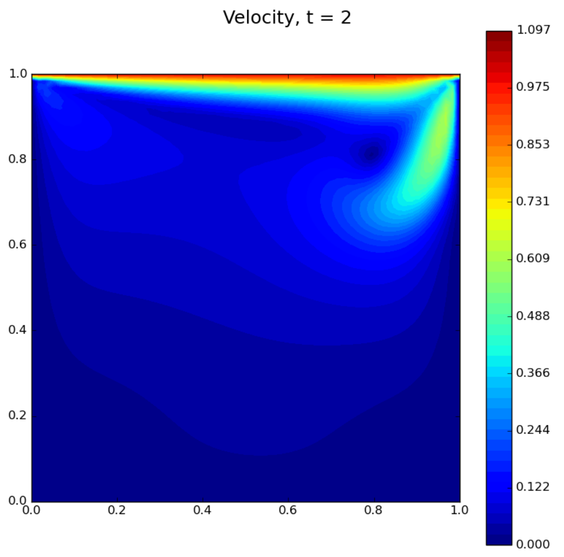

Figure C.39: Velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 2

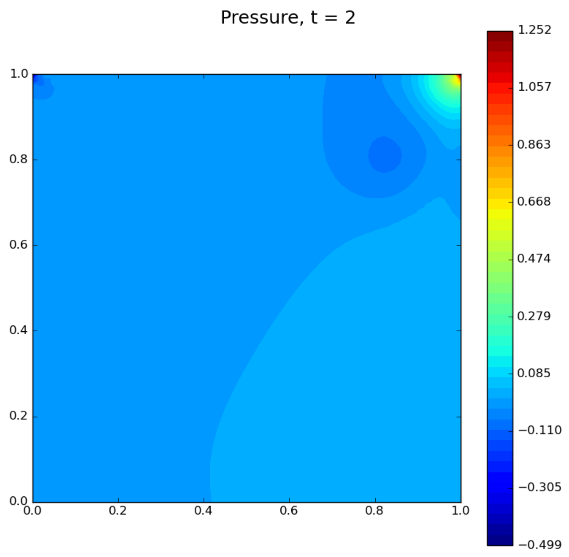

Figure C.40:
Pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 2

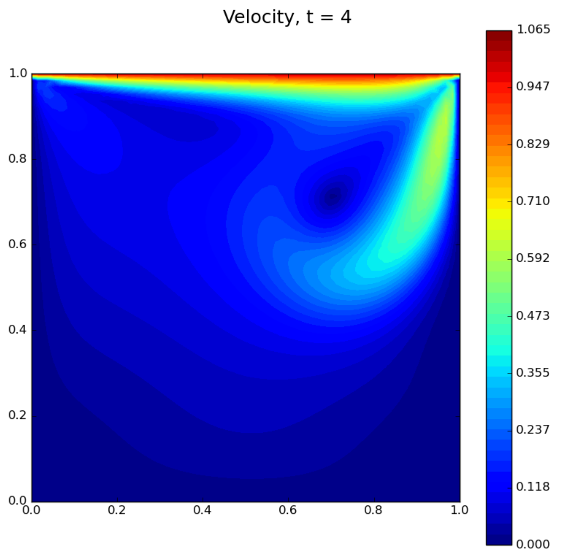

Figure C.41: Velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 4

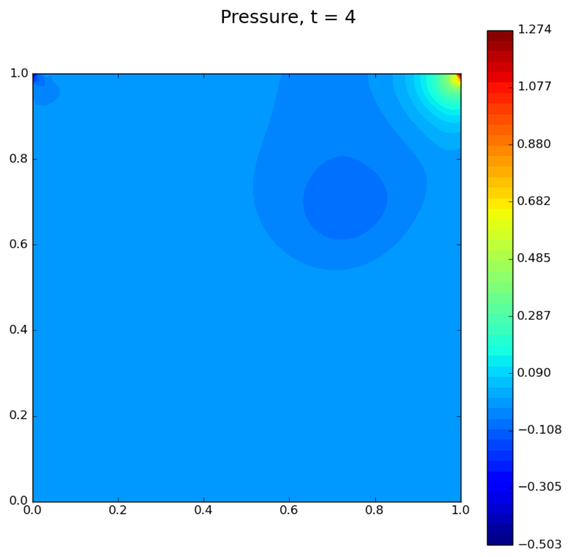

Figure C.42:
Pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 4

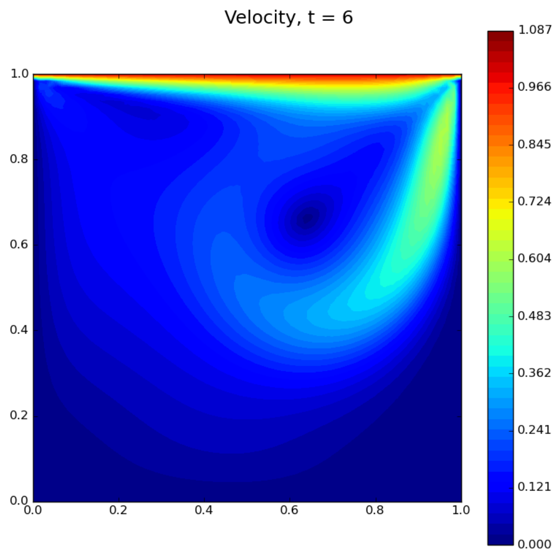

Figure C.43: Velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 6

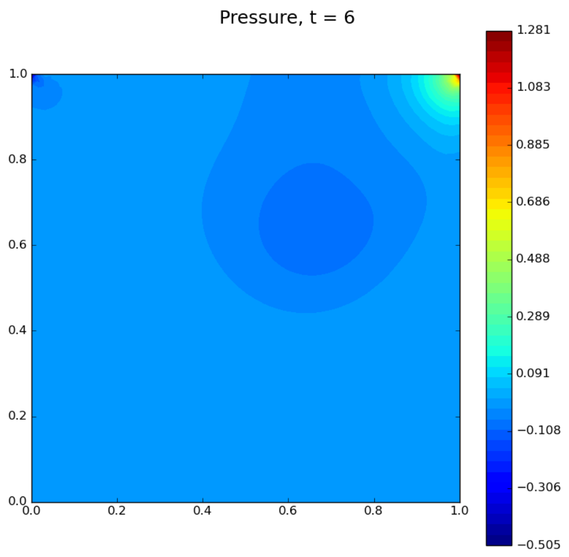

Figure C.44: Pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 6

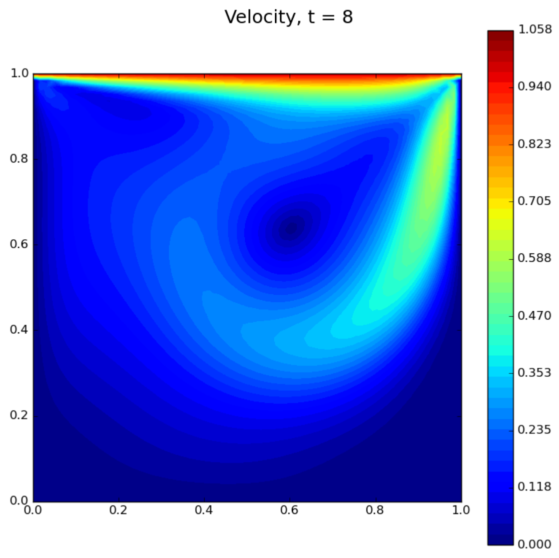

Figure C.45: Velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 8

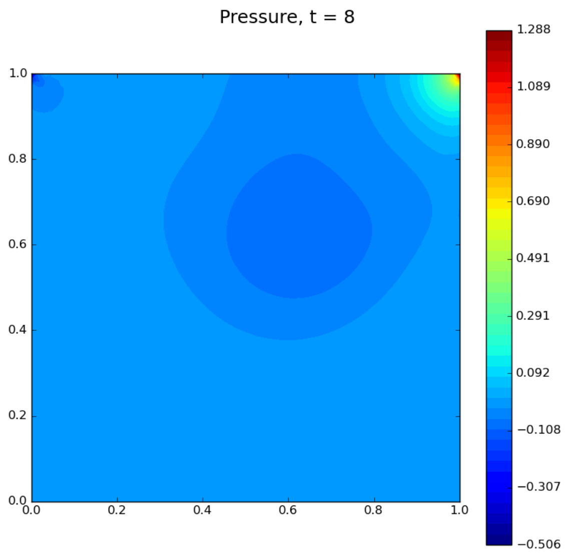

Figure C.46: Pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 8

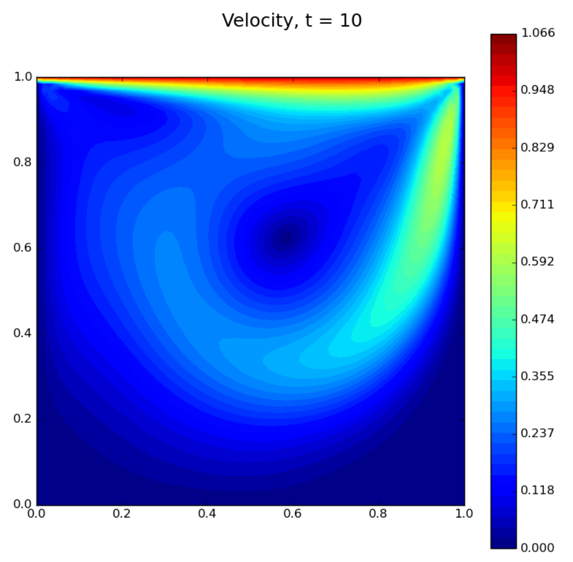

Figure C.47:
Velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 10

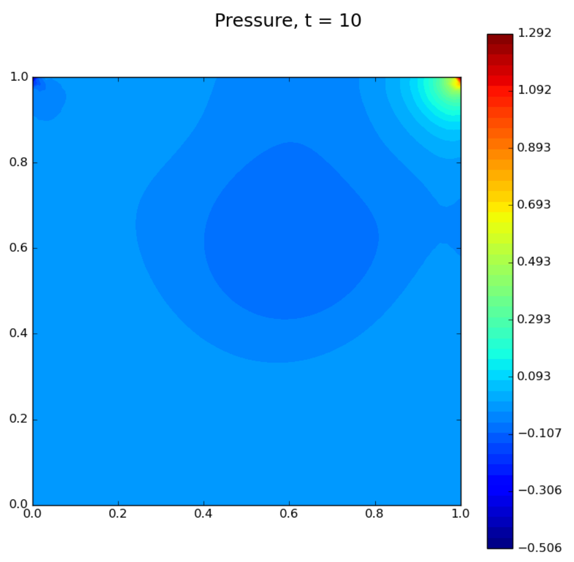

Figure C.48:
Pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 10

C.1.2. POD

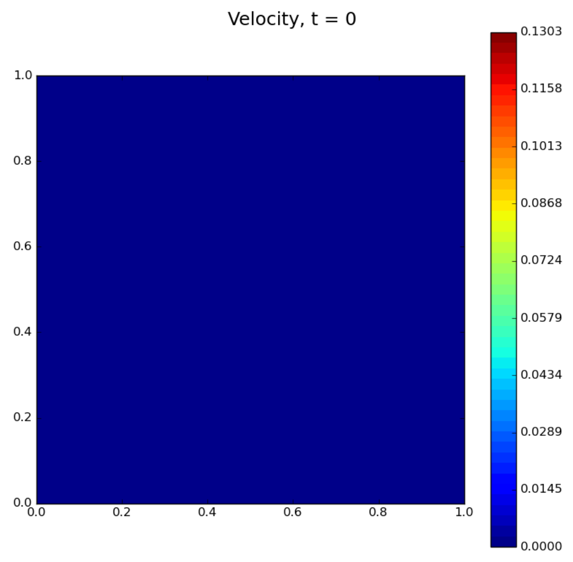

Figure C.49:
POD velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 0

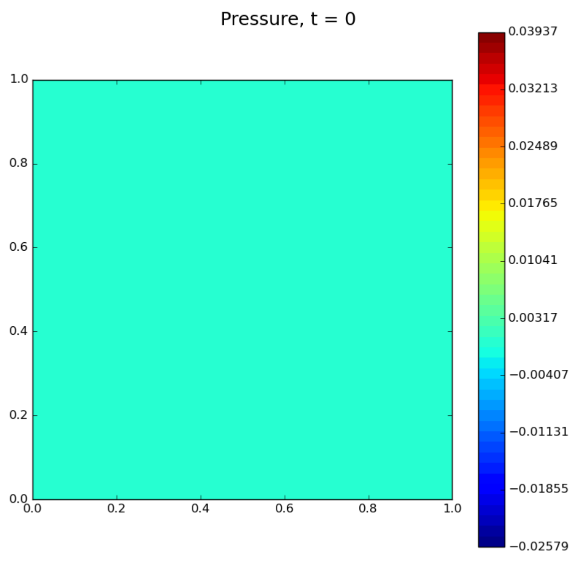

Figure C.50:
POD pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 0

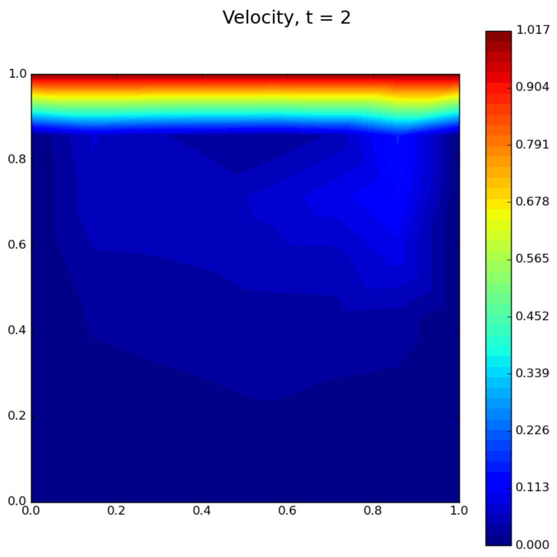

Figure C.51:
POD velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 2

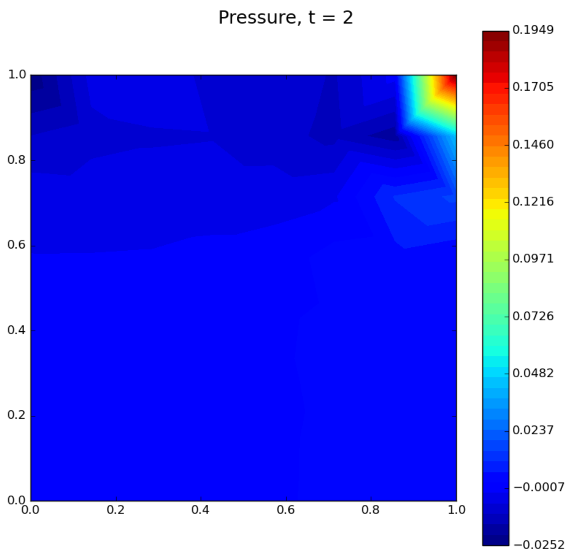

Figure C.52:
POD pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 2

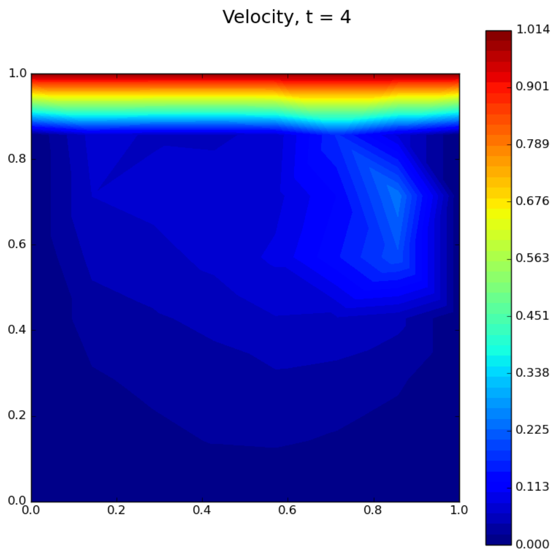

Figure C.53:
POD velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 4

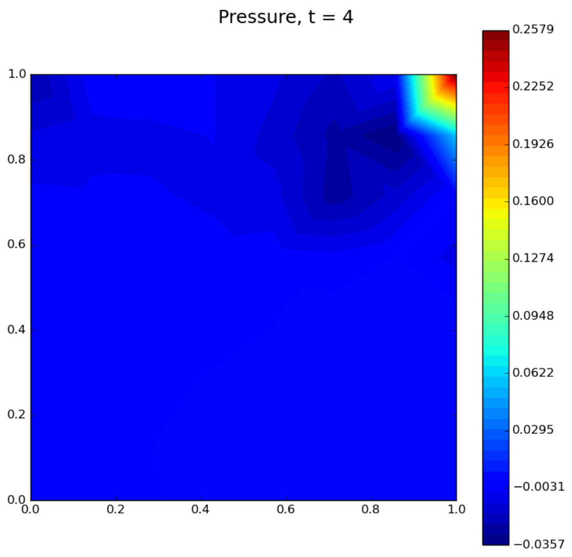

Figure C.54:
POD pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 4

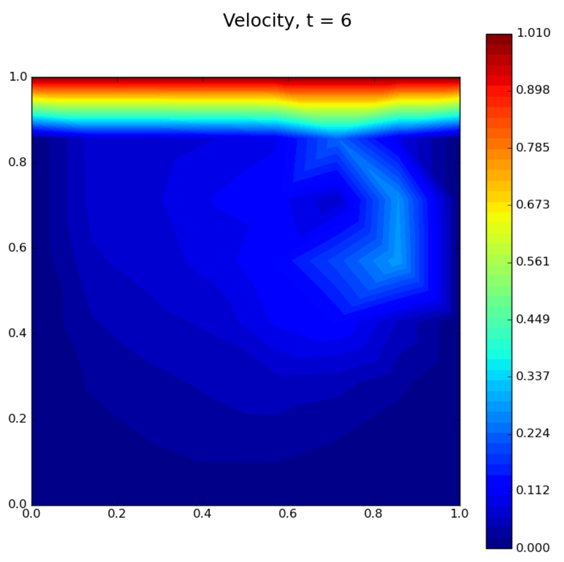

Figure C.55:
POD velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 6

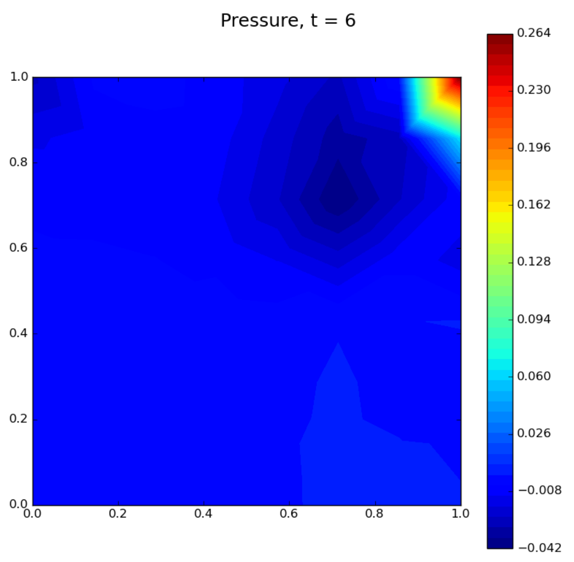

Figure C.56:
POD pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 6

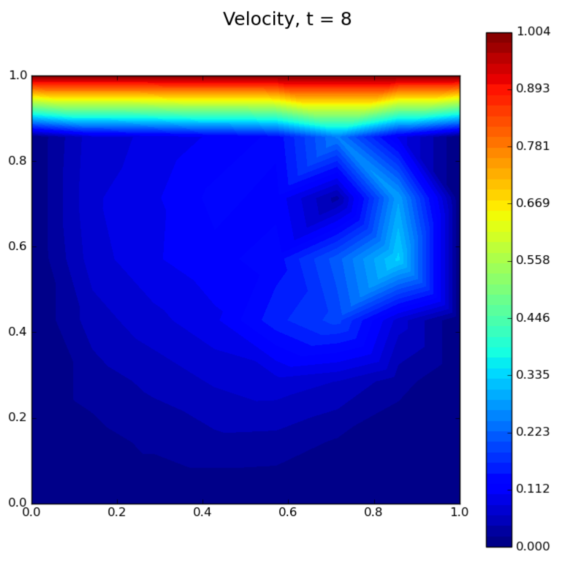

Figure C.57:
POD velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 8

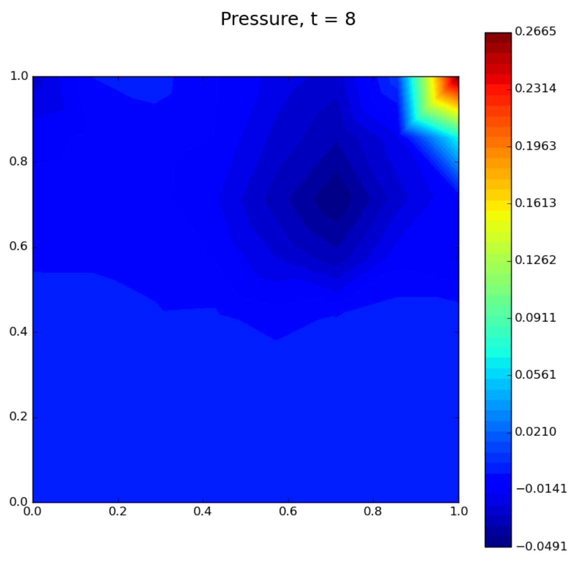

Figure C.58:
POD pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 8

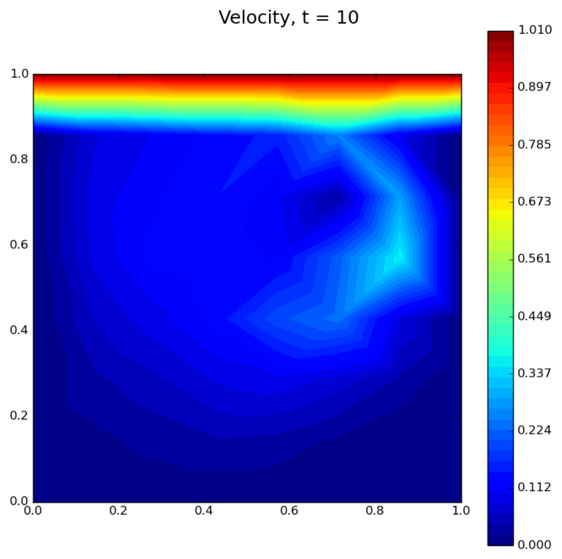

Figure C.59:
POD velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 10

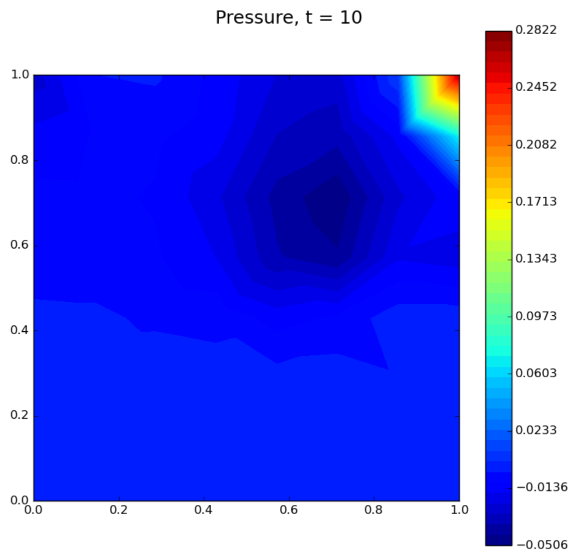

Figure C.60:
POD pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 10

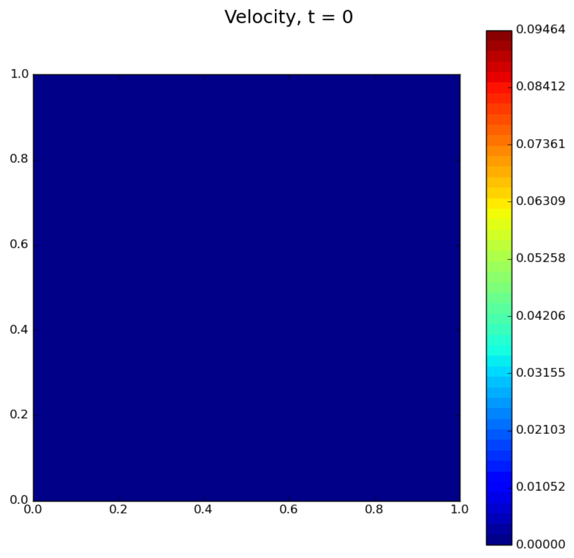

Figure C.61:
POD velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 0

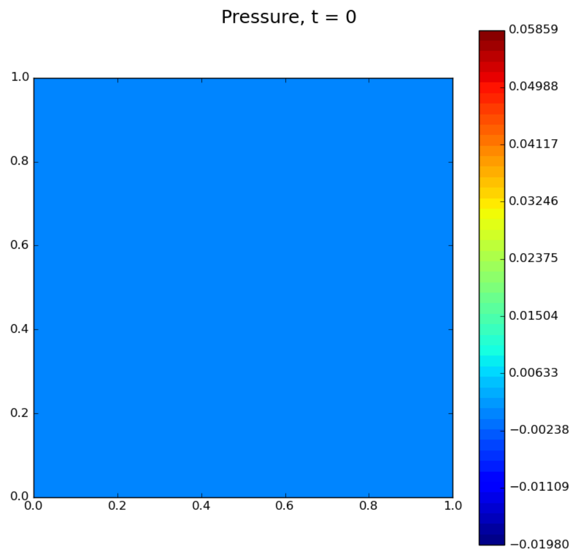

Figure C.62:
POD pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 0

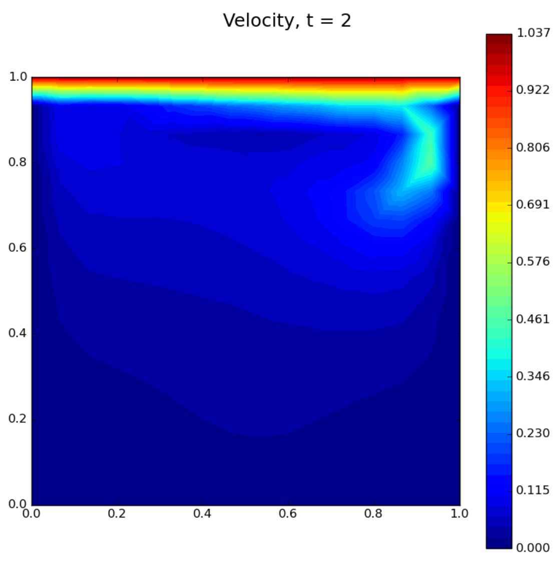

Figure C.63:
POD velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 2

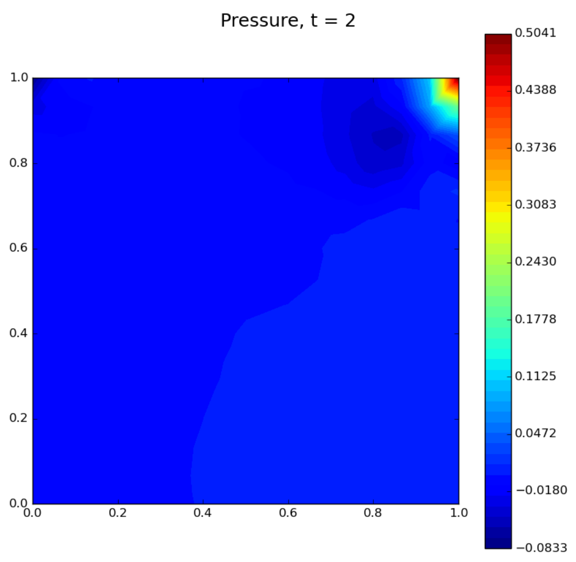

Figure C.64:
POD pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 2

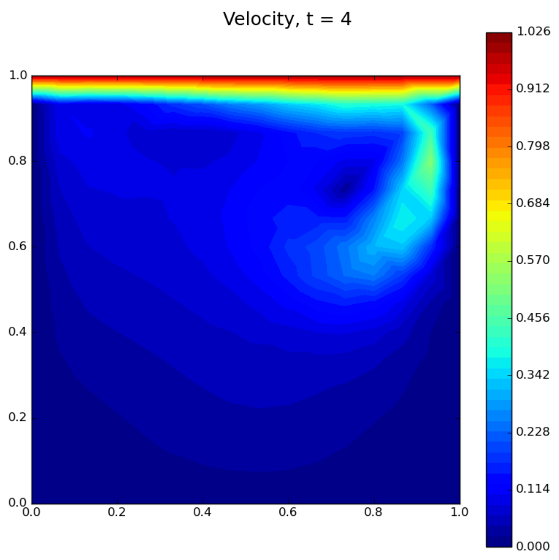

Figure C.65:
POD velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 4

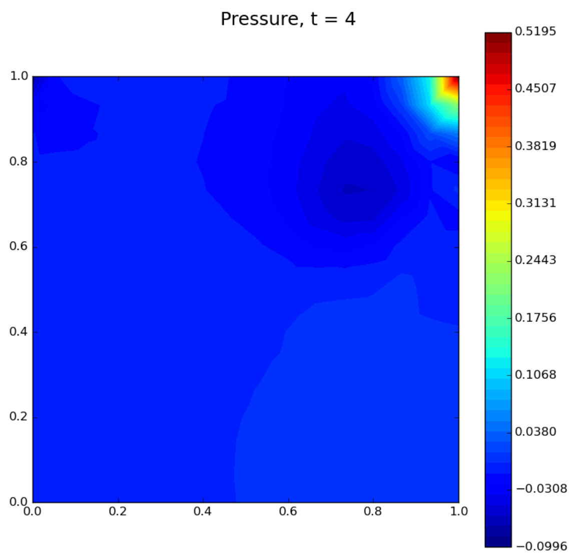

Figure C.66:
POD pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 4

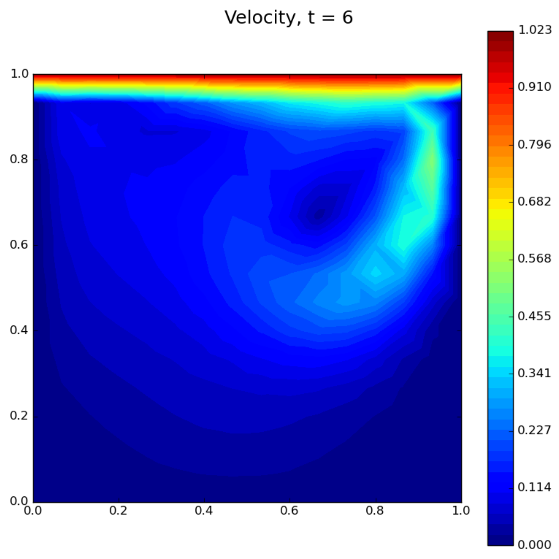

Figure C.67:
POD velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 6

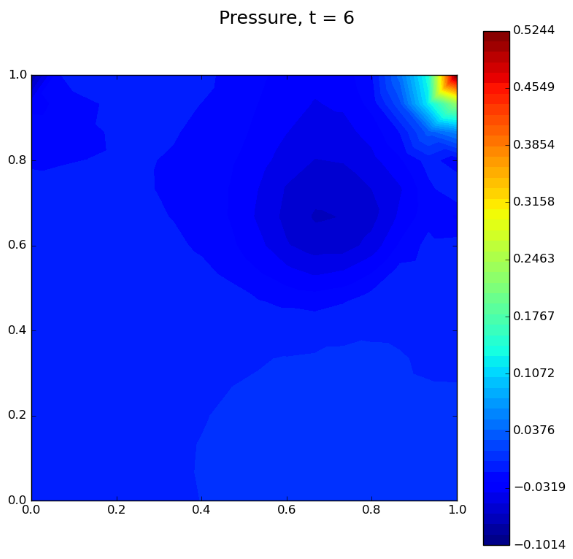

Figure C.68:
POD pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 6

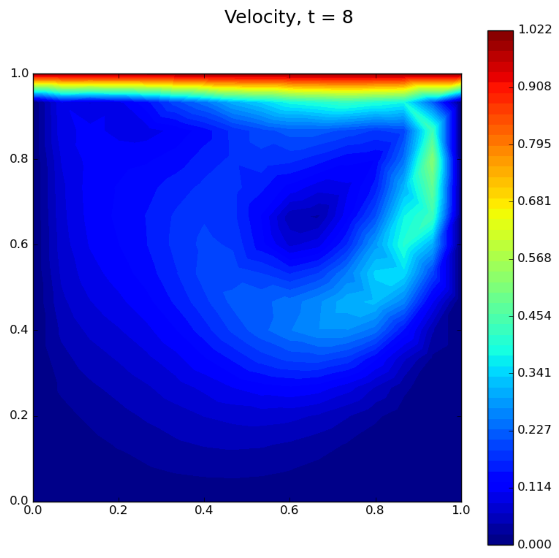

Figure C.69:
POD velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 8

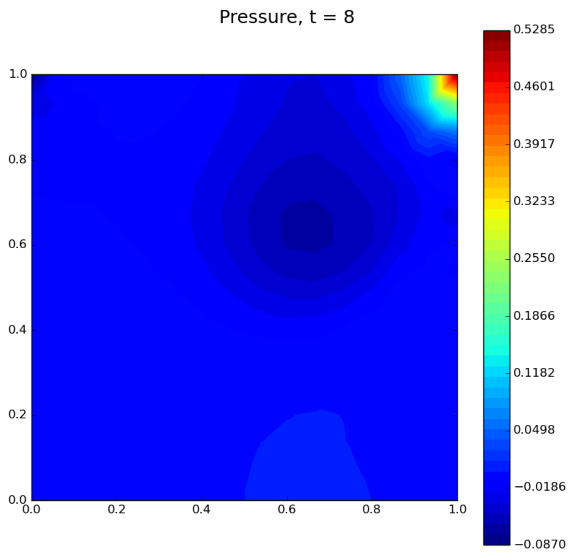

Figure C.70:
POD pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 8

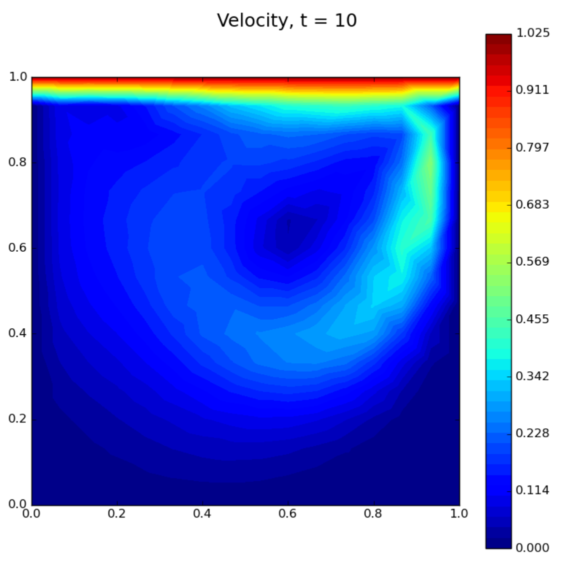

Figure C.71:
POD velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 10

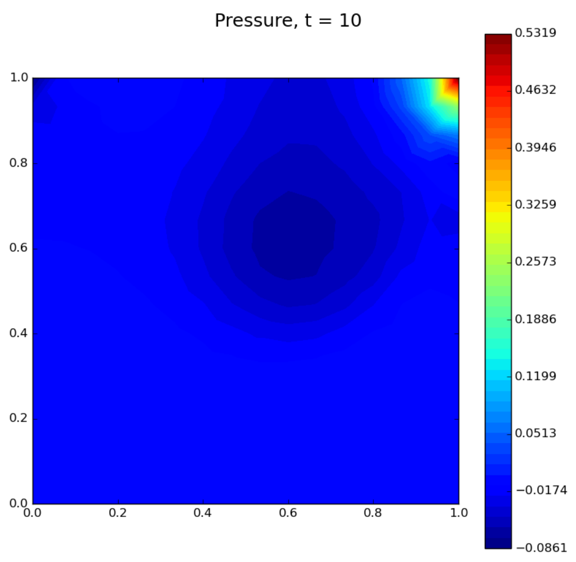

Figure C.72:
POD pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 10

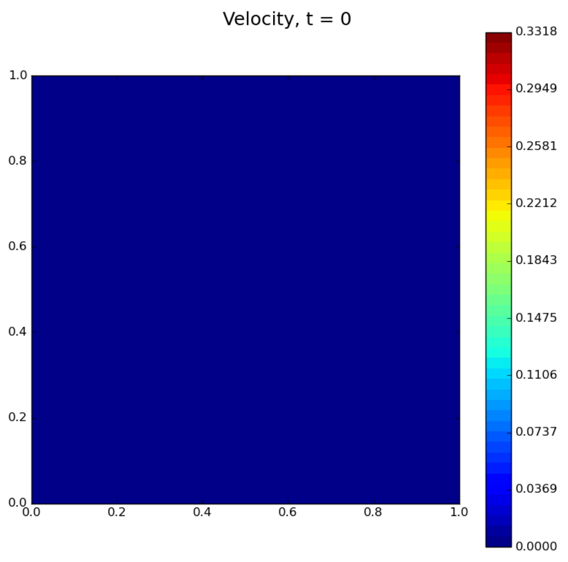

Figure C.73:
POD velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 0

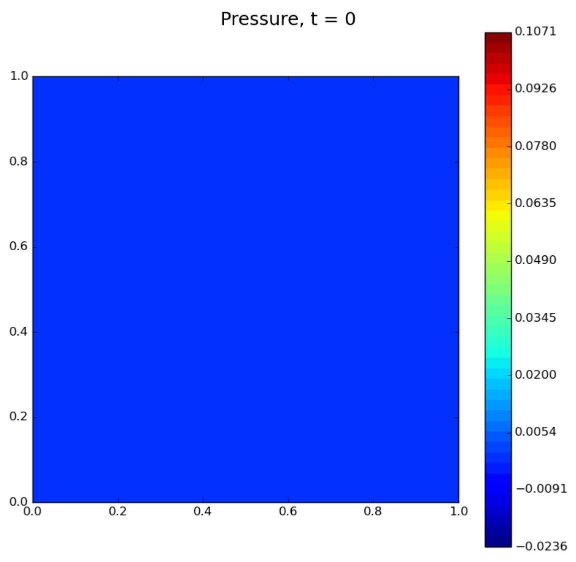

Figure C.74:
POD pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 0

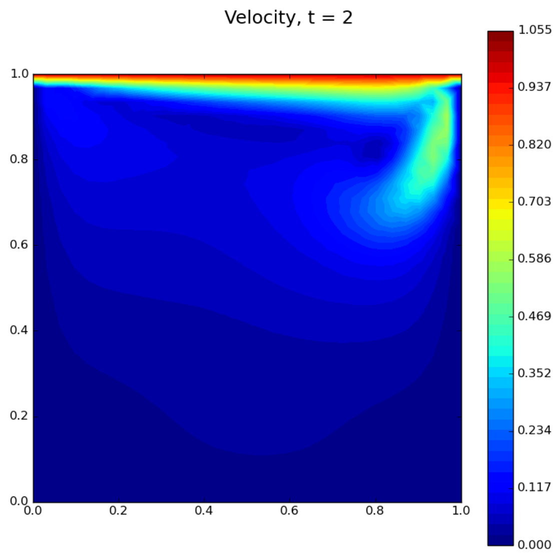

Figure C.75:
POD velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 2

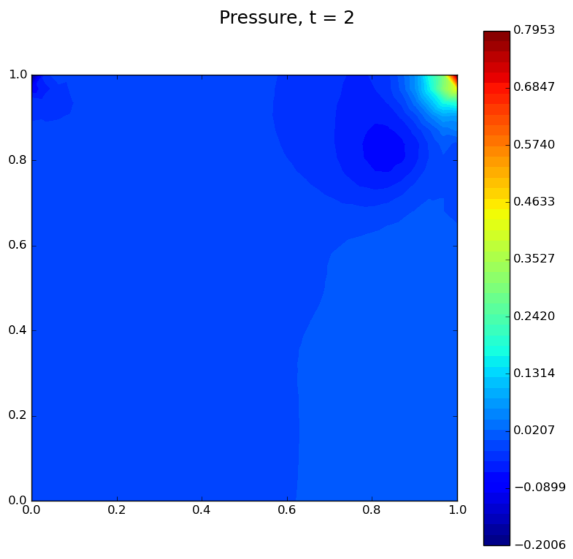

Figure C.76:
POD pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 2

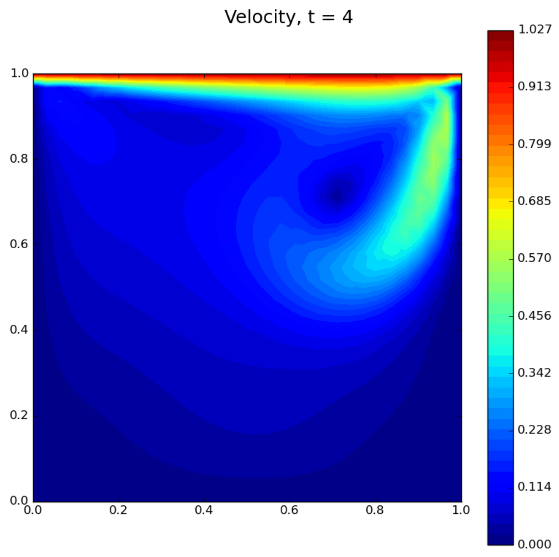

Figure C.77:
POD velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 4

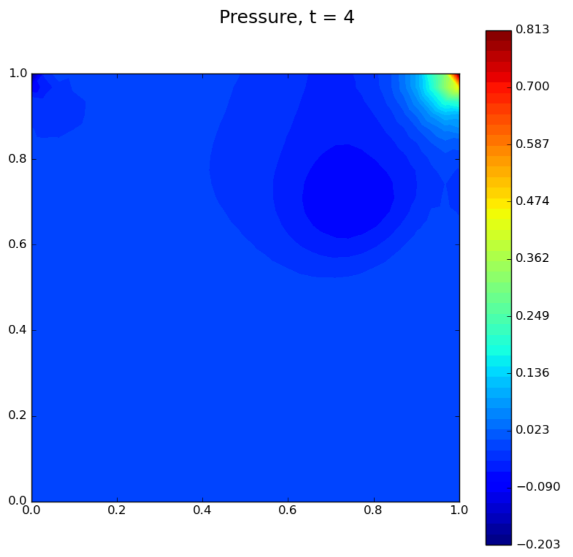

Figure C.78:
POD pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 4

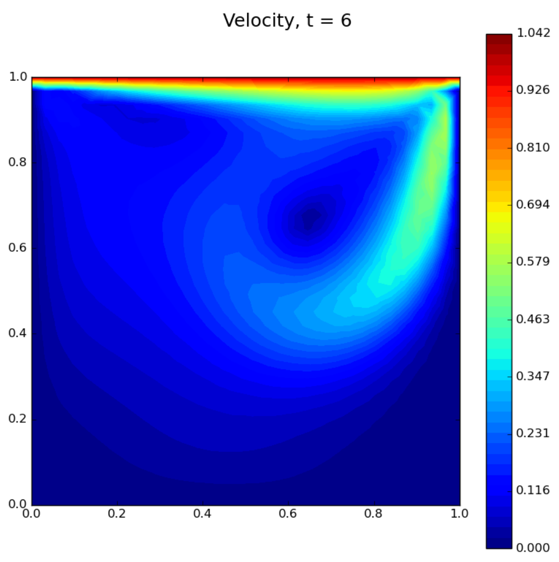

Figure C.79:
POD velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 6

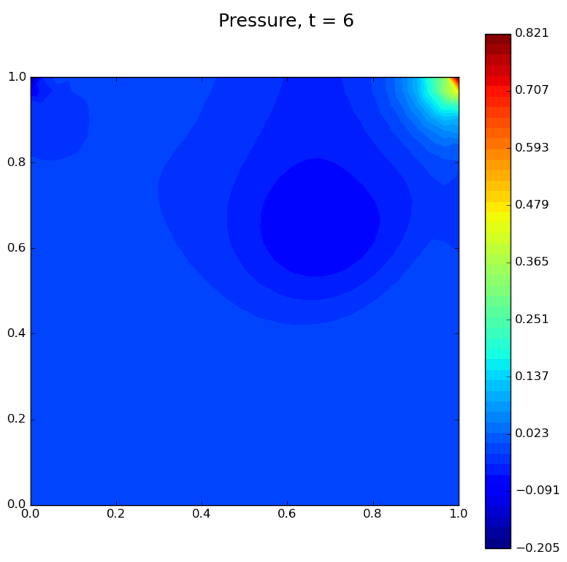

Figure C.80:
POD pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 6

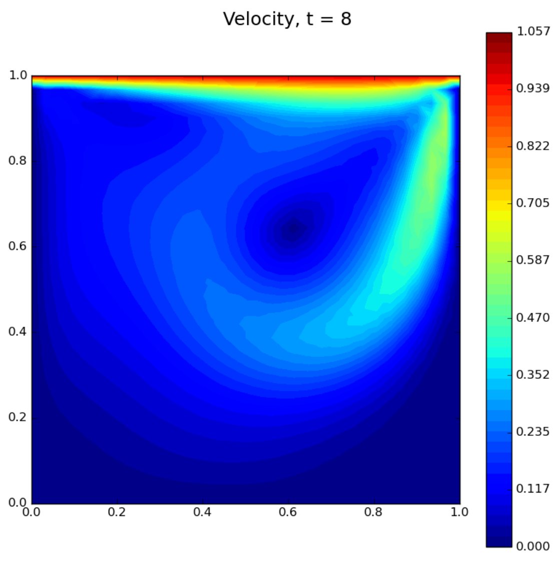

Figure C.81:
POD velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 8

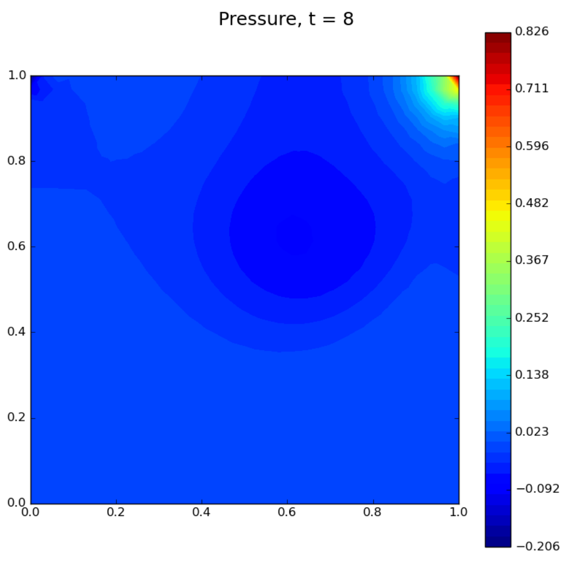

Figure C.82:
POD pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 8

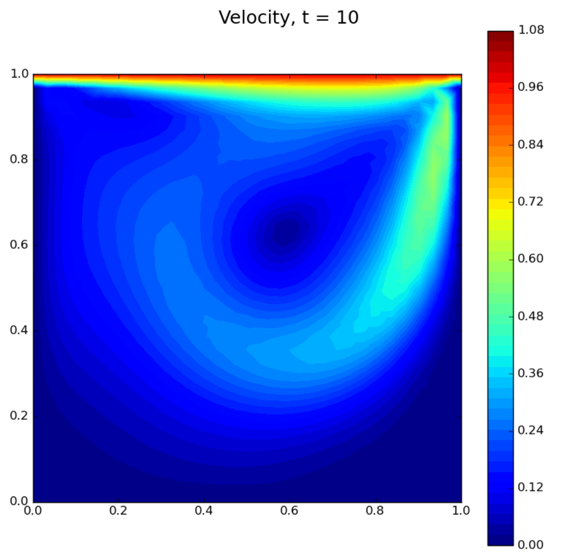

Figure C.83:
POD velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 10

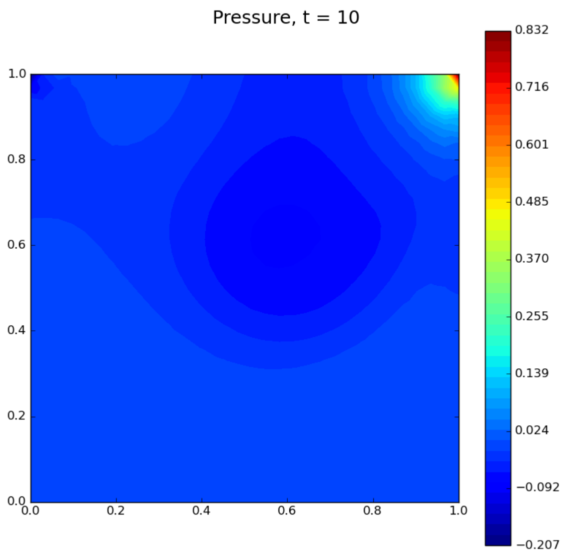

Figure C.84:
POD pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 10

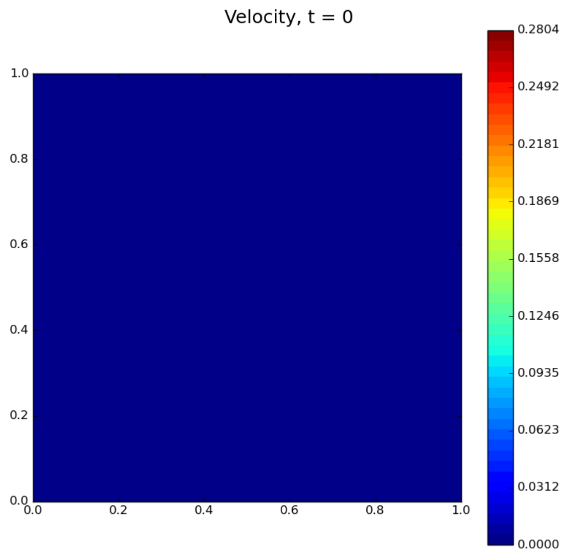

Figure C.85:
POD velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 0

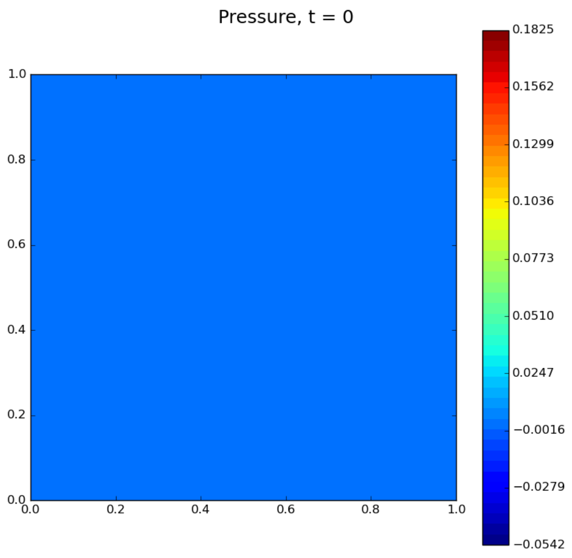

Figure C.86:
POD pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 0

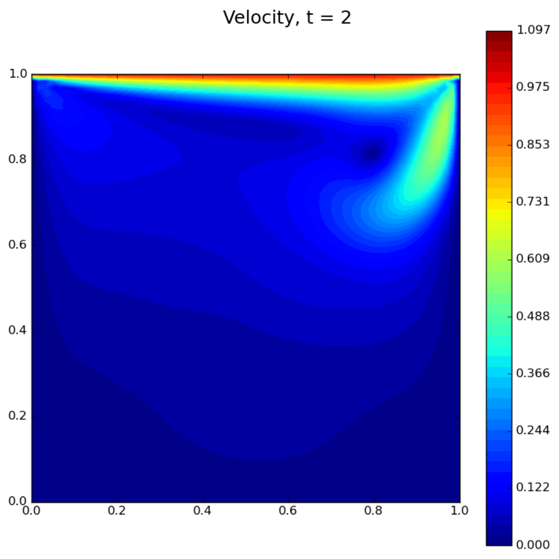

Figure C.87:
POD velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 2

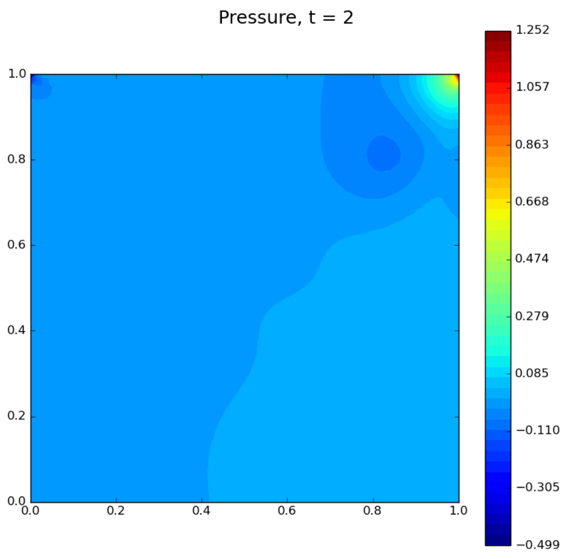

Figure C.88:
POD pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 2

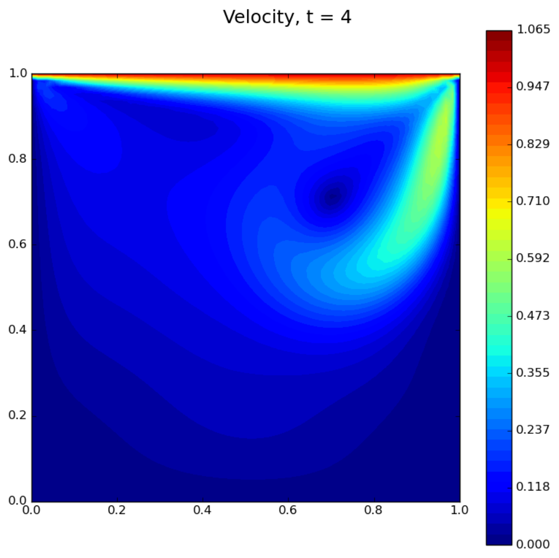

Figure C.89:
POD velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 4

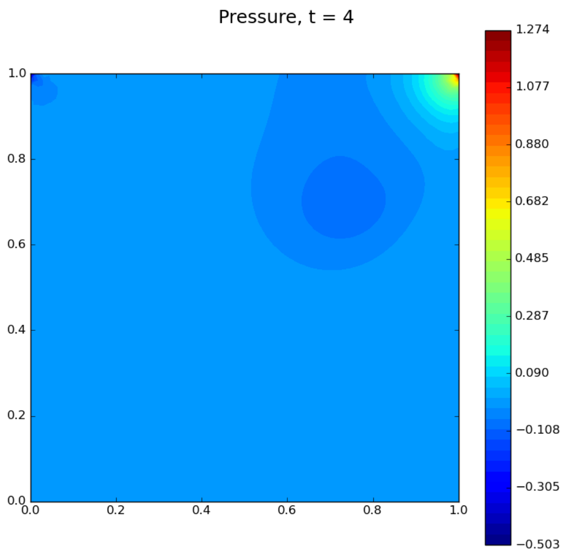

Figure C.90:
POD pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 4

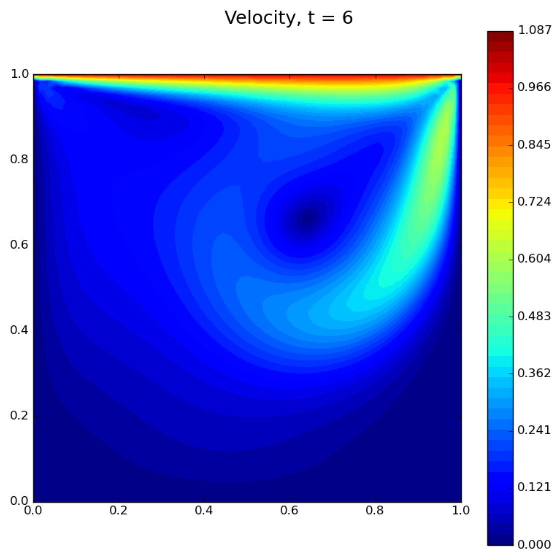

Figure C.91:
POD velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 6

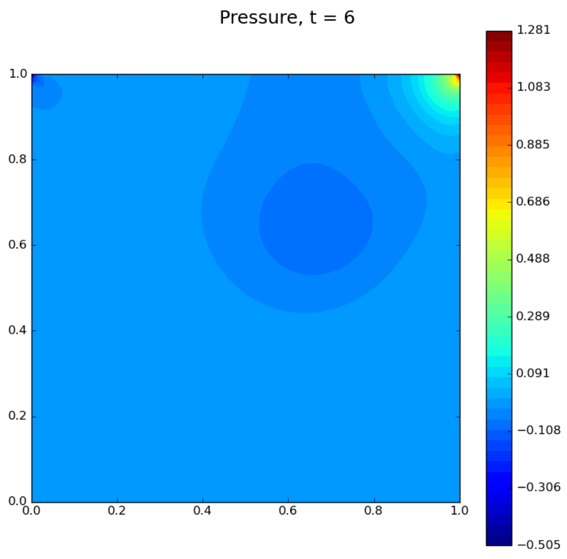

Figure C.92:
POD pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 6

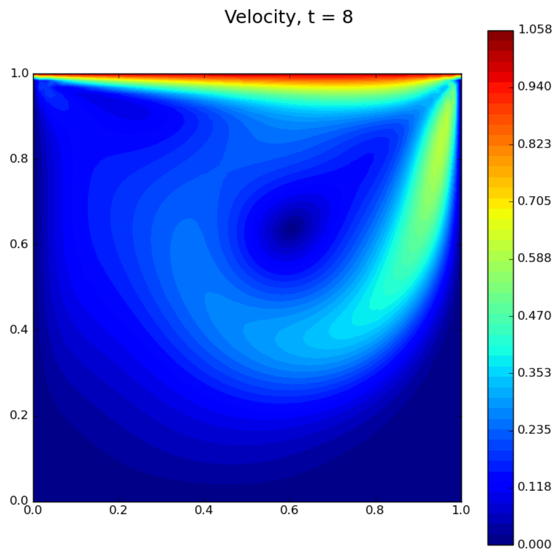

Figure C.93:
POD velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 8

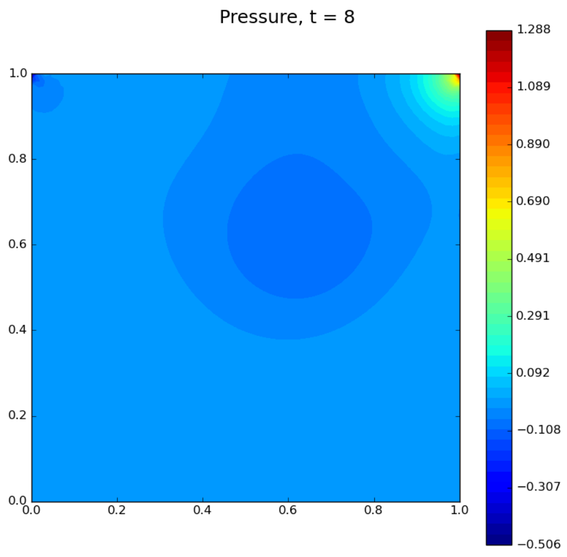

Figure C.94:
POD pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 8

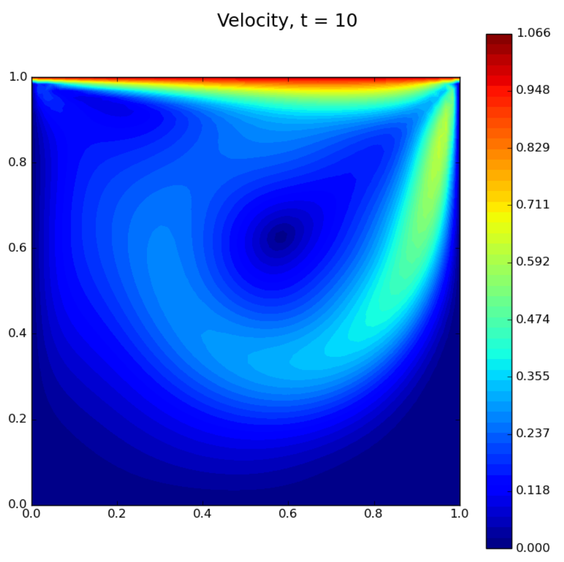

Figure C.95:
POD velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 10

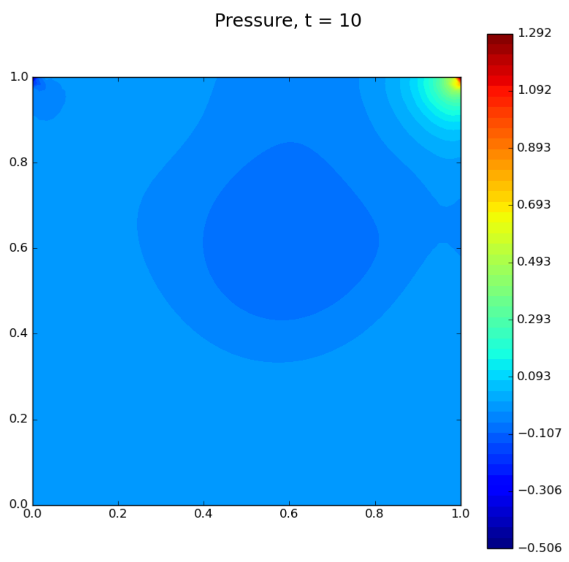

Figure C.96:
POD pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 10

C.1.3. Autoencoder

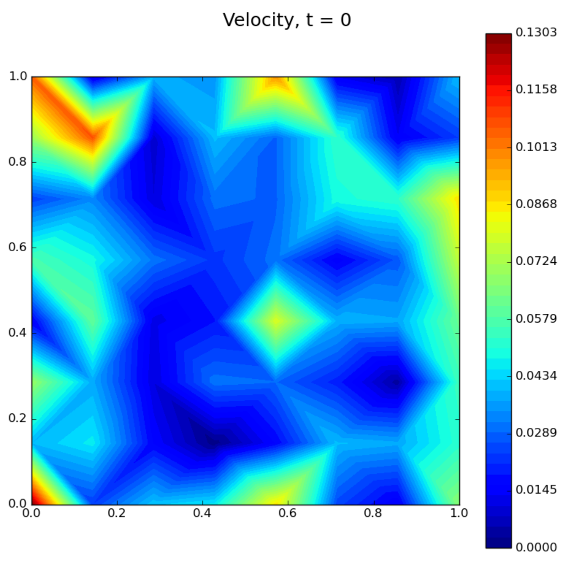

Figure C.97:
Autoencoder velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 0

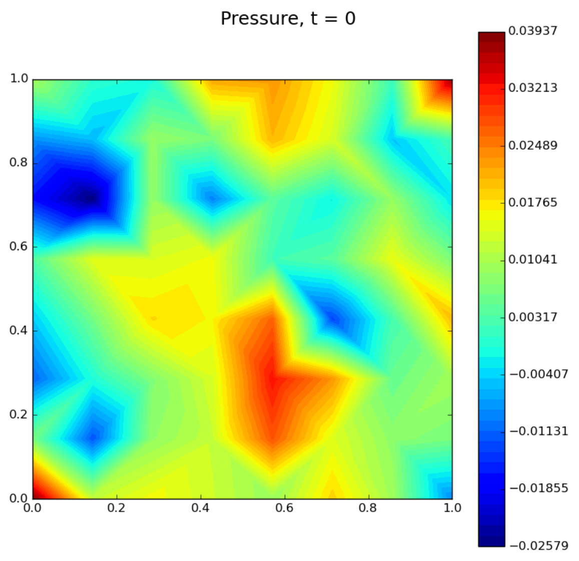

Figure C.98: Autoencoder
pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 0

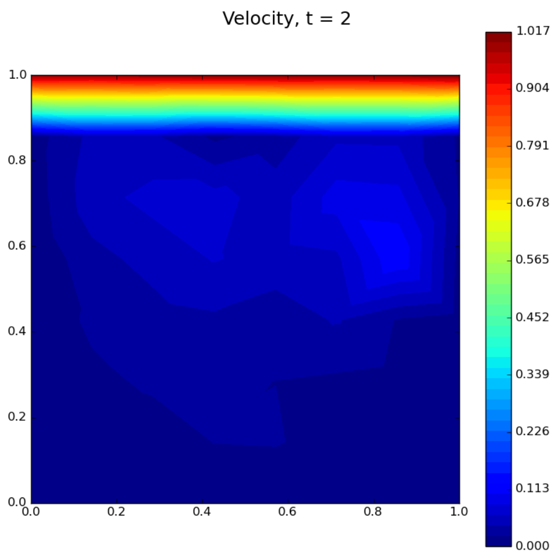

Figure C.99:
Autoencoder velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 2

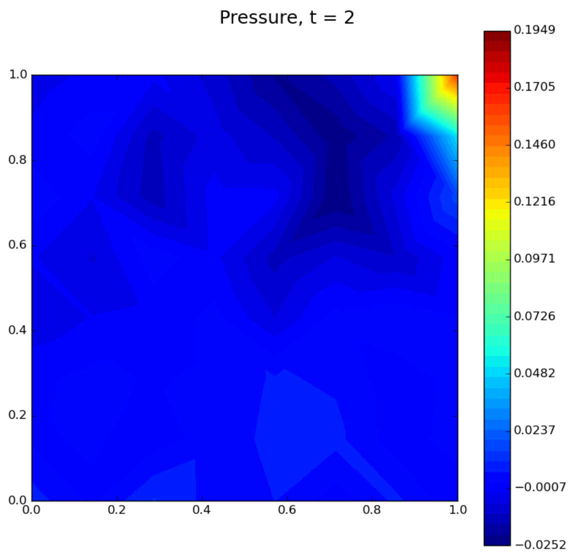

Figure C.100: Autoencoder
pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 2

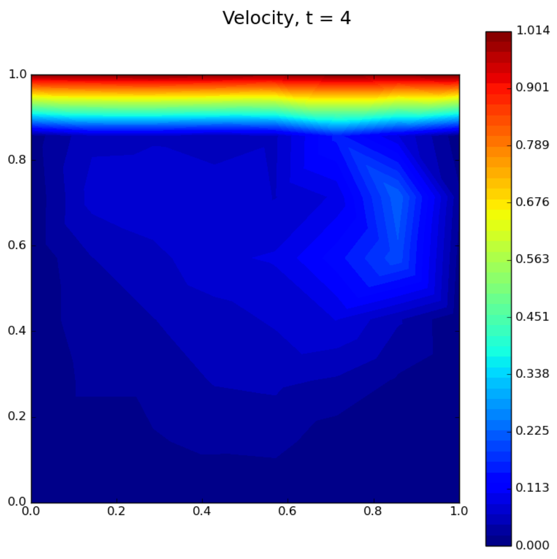

Figure C.101:
Autoencoder velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 4

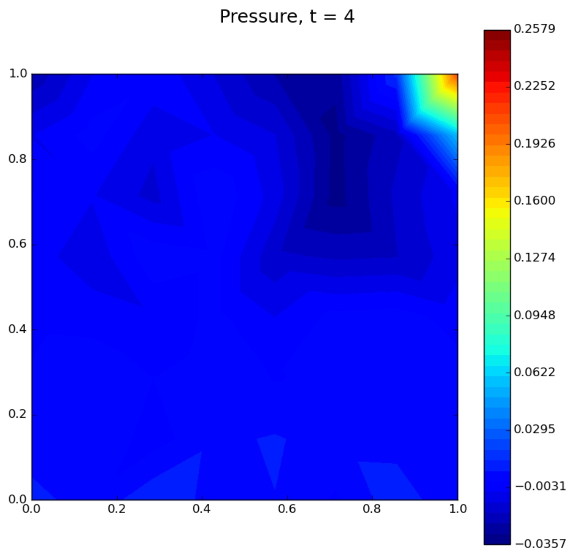

Figure C.102: Autoencoder
pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 4

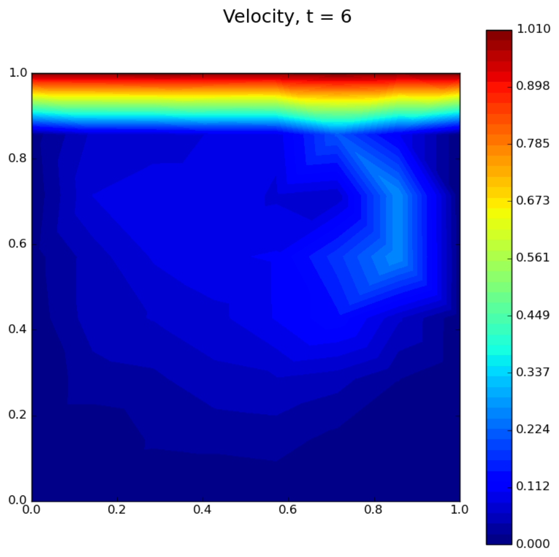

Figure C.103:
Autoencoder velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 6

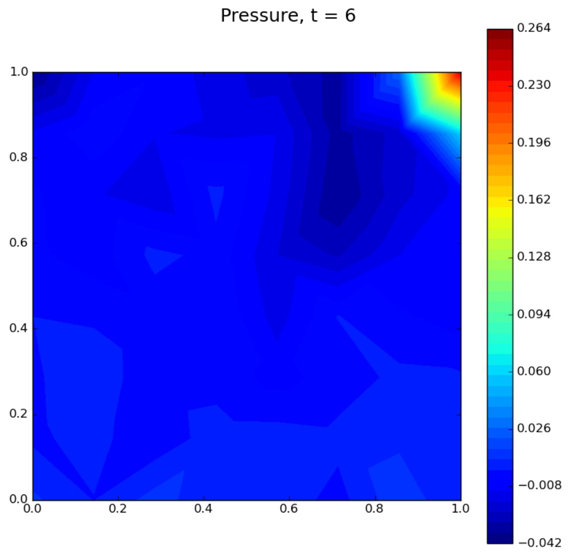

Figure C.104: Autoencoder pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 6

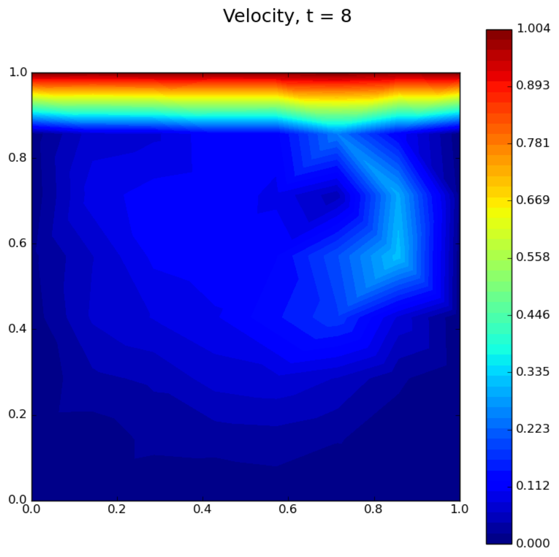

Figure C.105:
Autoencoder velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 8

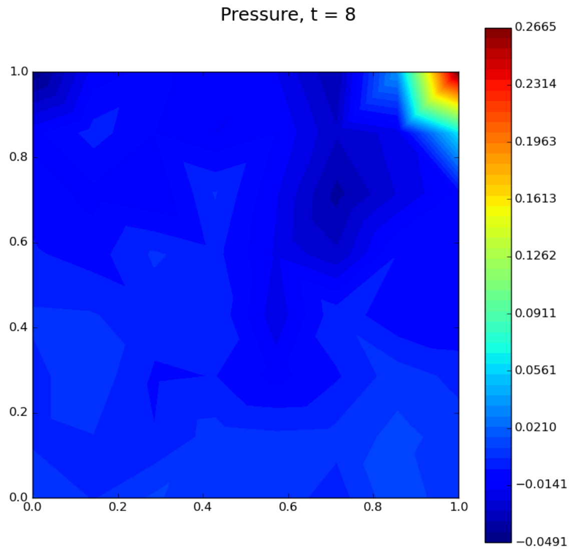

Figure C.106: Autoencoder pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 8

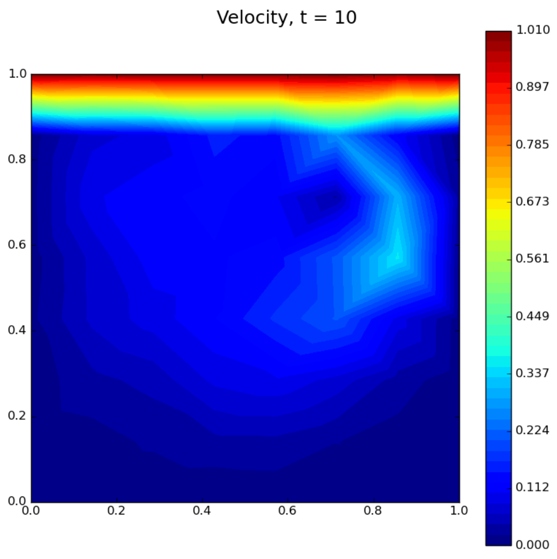

Figure C.107: Autoencoder velocity field for $N_H = 8 \times 8$ and Re = 500 for t = 10

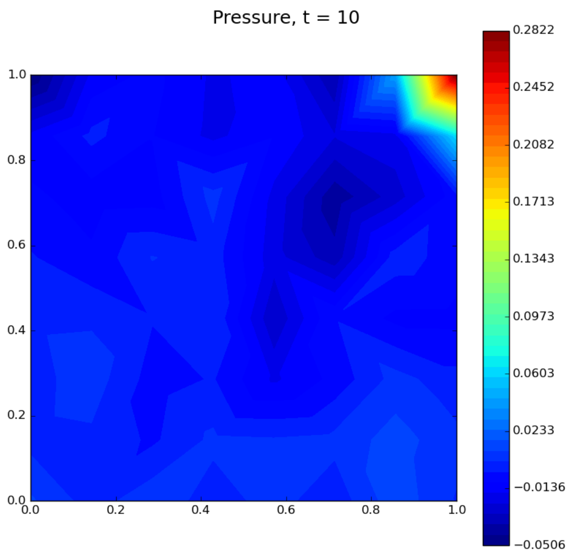

Figure C.108: Autoencoder pressure field for $N_H = 8 \times 8$ and Re = 500 for t = 10

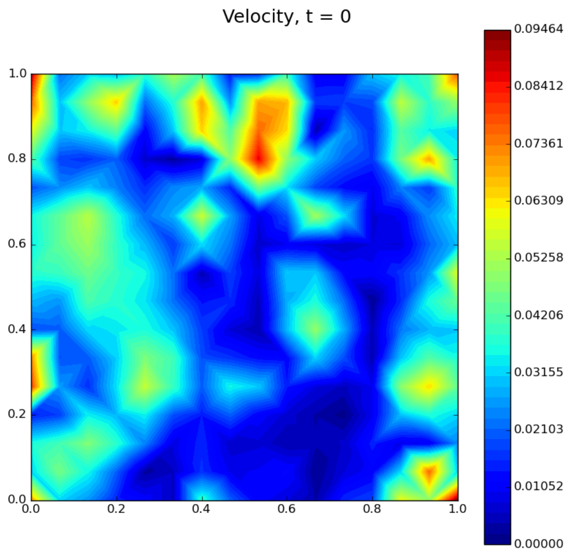

Figure C.109: Autoencoder velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 0

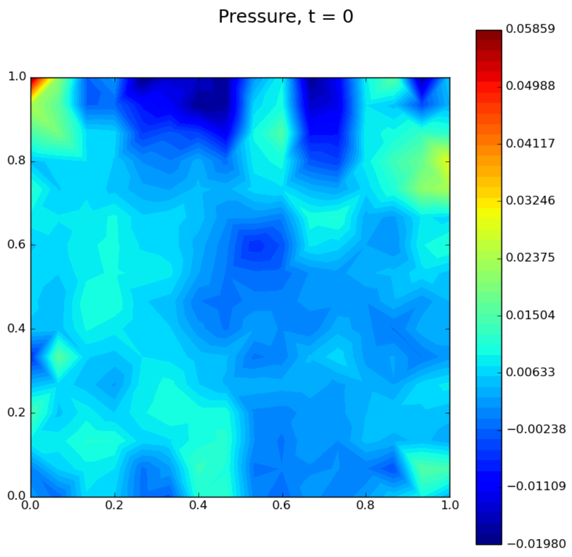

Figure C.110: Autoencoder pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 0

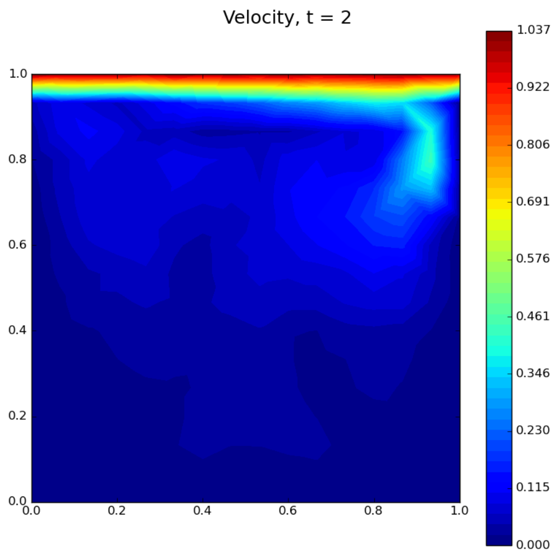

Figure C.111: Autoencoder velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 2

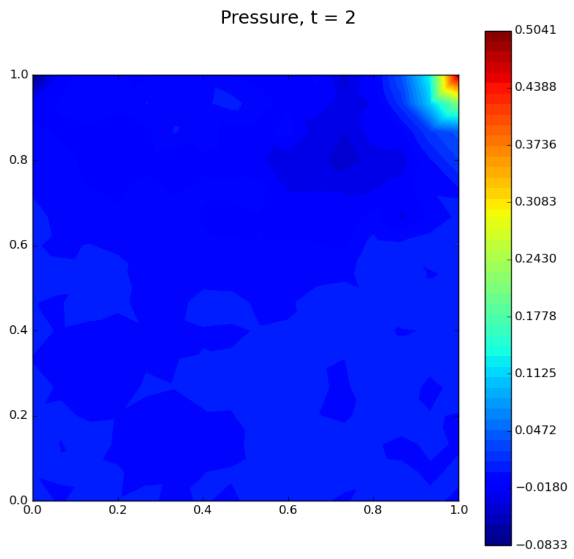

Figure C.112: Autoencoder pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 2

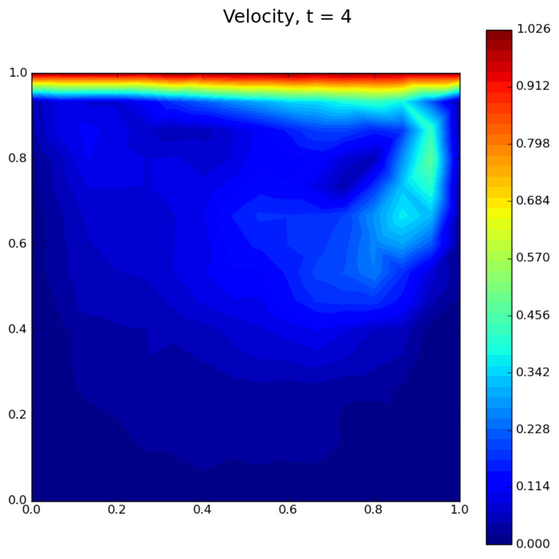

Figure C.113: Autoencoder velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 4

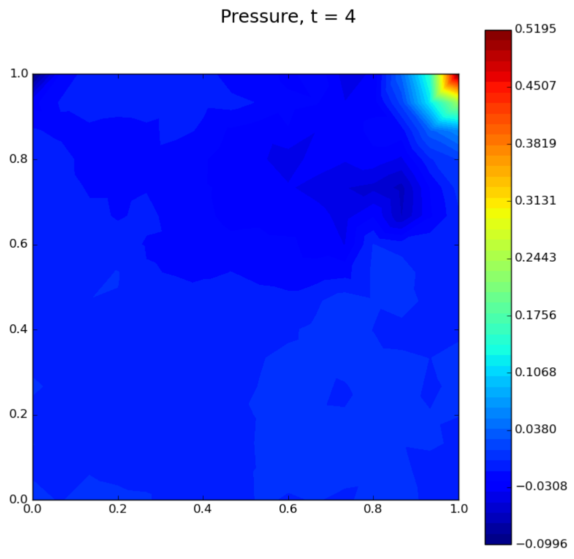

Figure C.114: Autoencoder pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 4

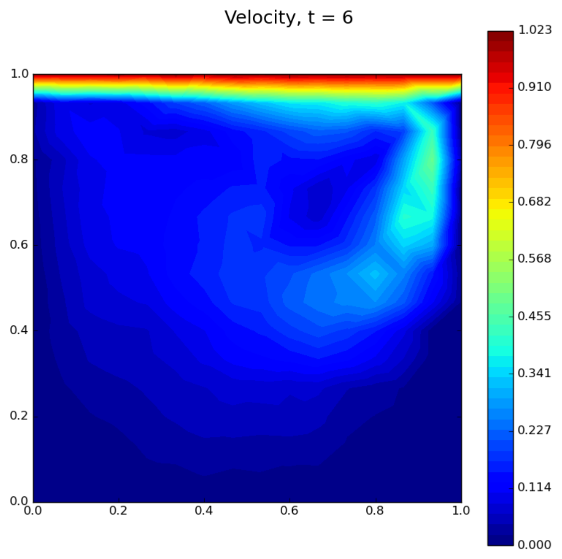

Figure C.115: Autoencoder velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 6

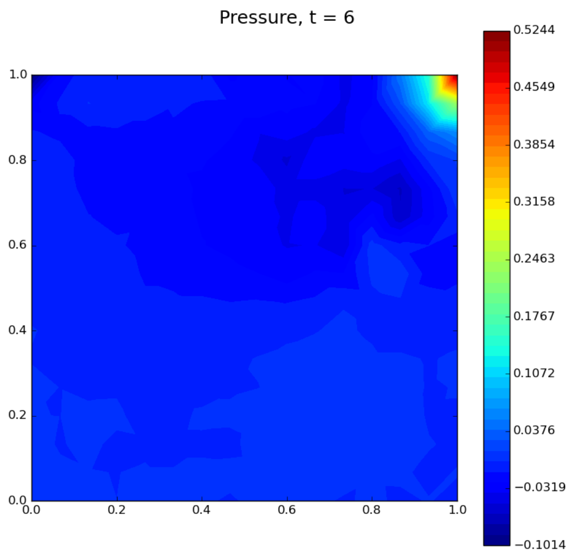

Figure C.116: Autoencoder pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 6

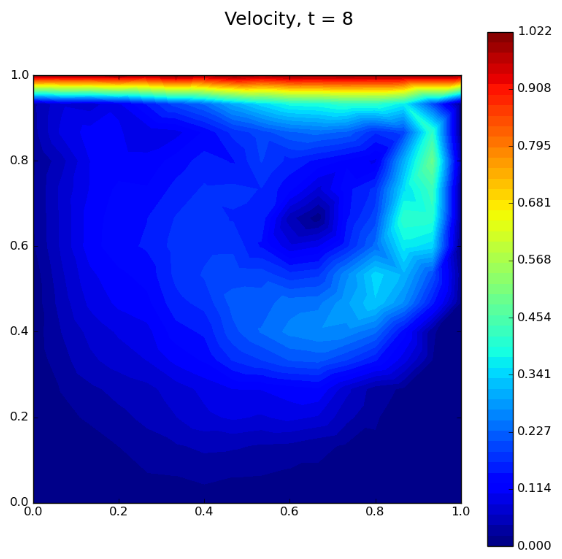

Figure C.117: Autoencoder velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 8

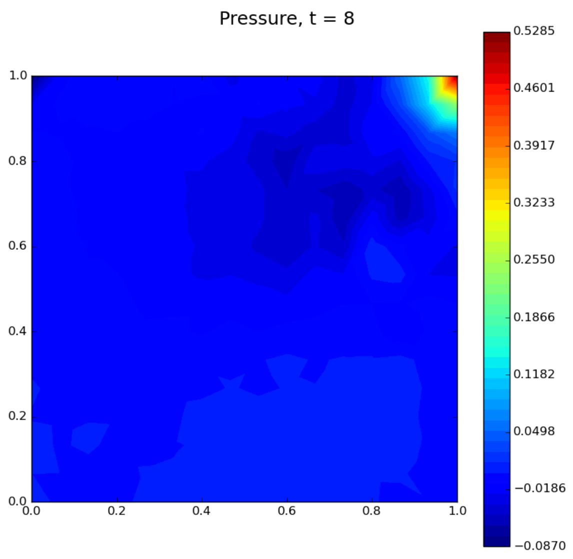

Figure C.118: Autoencoder pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 8

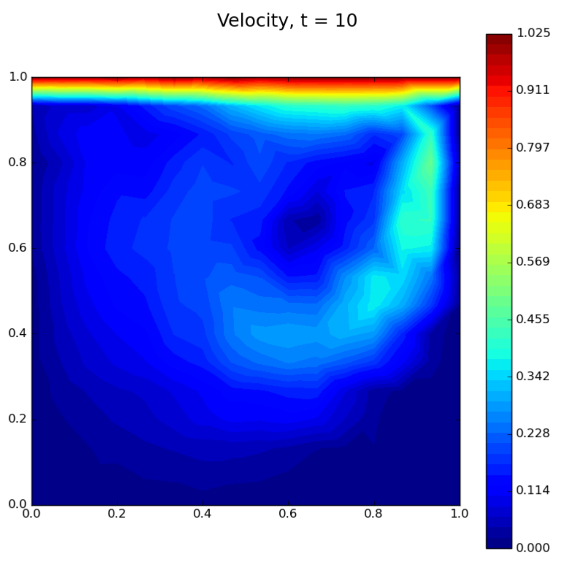

Figure C.119: Autoencoder
velocity field for $N_H = 16 \times 16$ and Re = 500 for t = 10

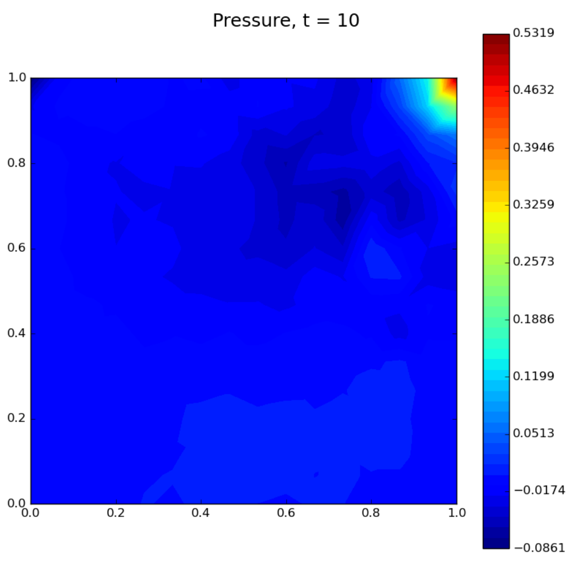

Figure C.120: Autoencoder
pressure field for $N_H = 16 \times 16$ and Re = 500 for t = 10

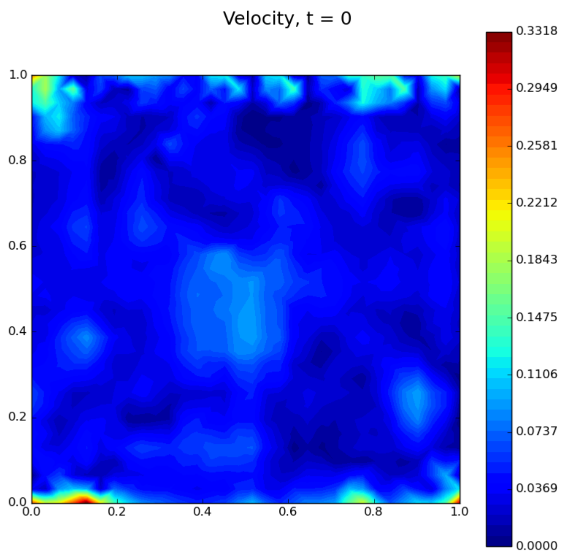

Figure C.121: Autoencoder
velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 0

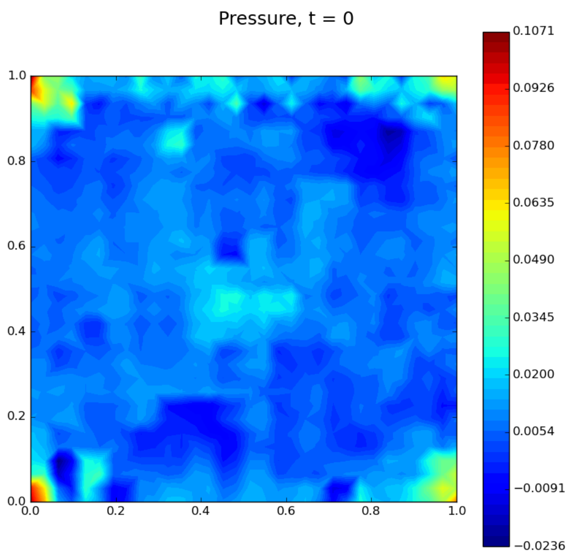

Figure C.122: Autoencoder
pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 0

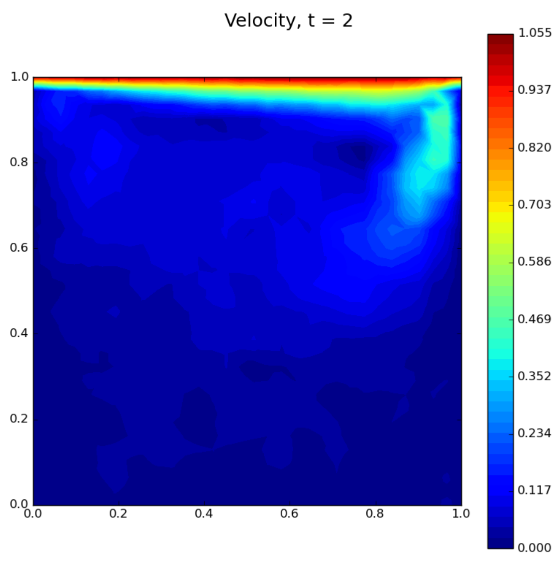

Figure C.123: Autoencoder velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 2

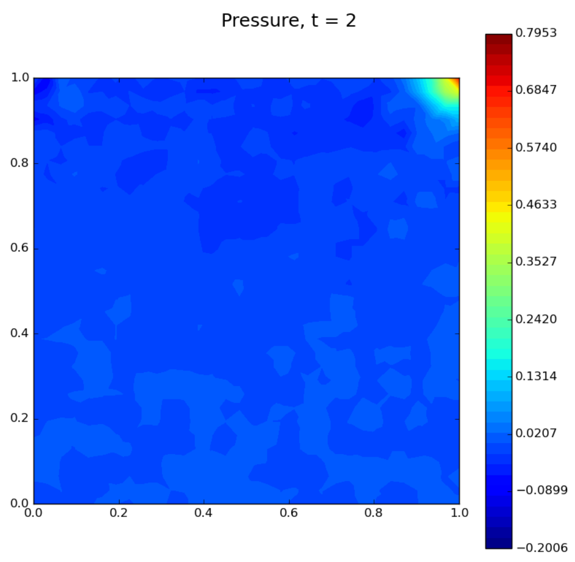

Figure C.124: Autoencoder pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 2

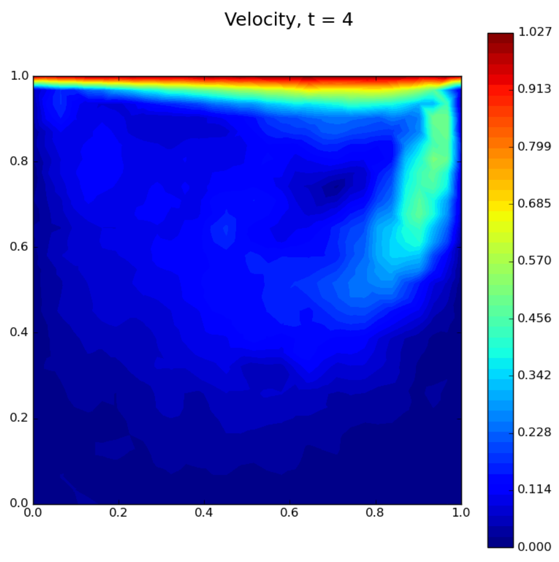

Figure C.125: Autoencoder velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 4

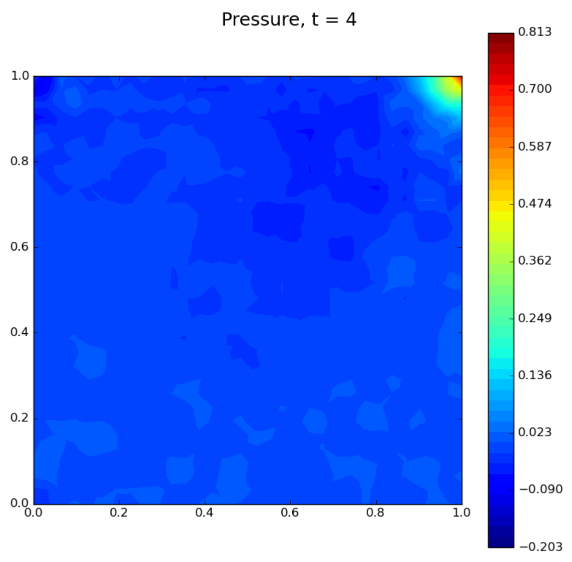

Figure C.126: Autoencoder pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 4

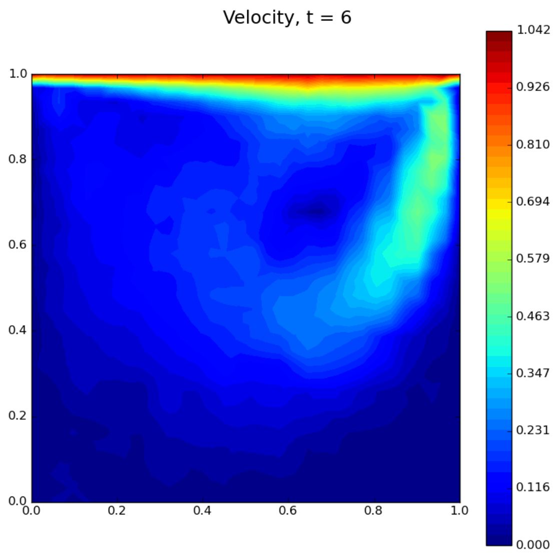

Figure C.127: Autoencoder
velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 6

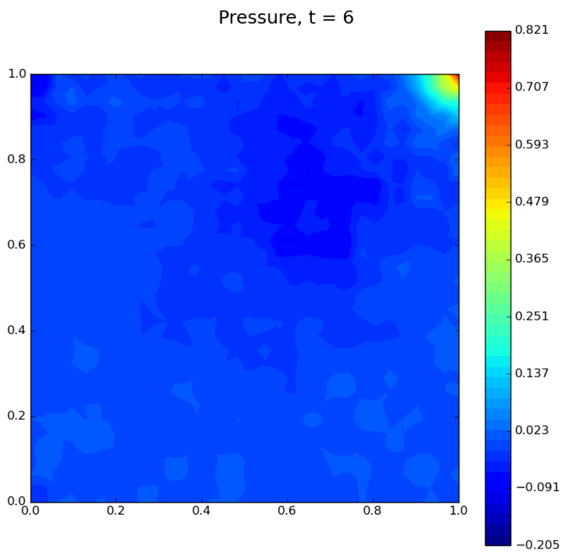

Figure C.128: Autoencoder
pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 6

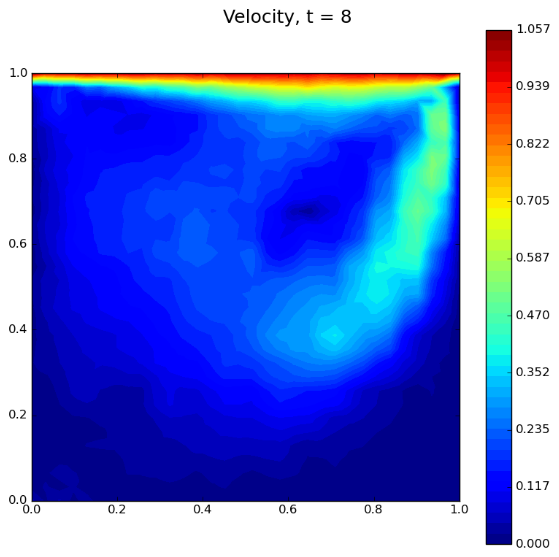

Figure C.129: Autoencoder
velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 8

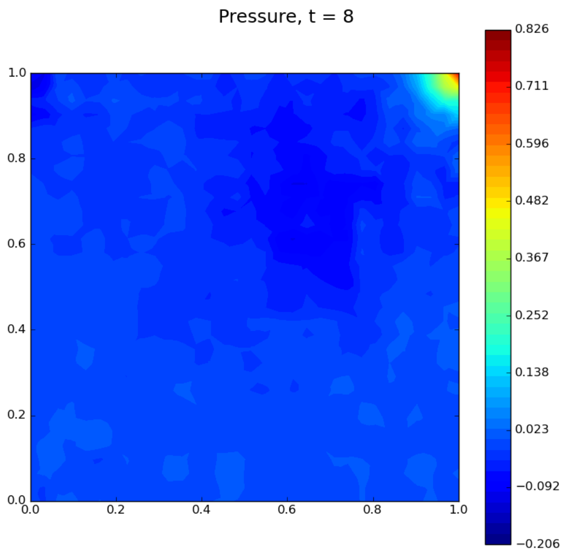

Figure C.130: Autoencoder
pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 8

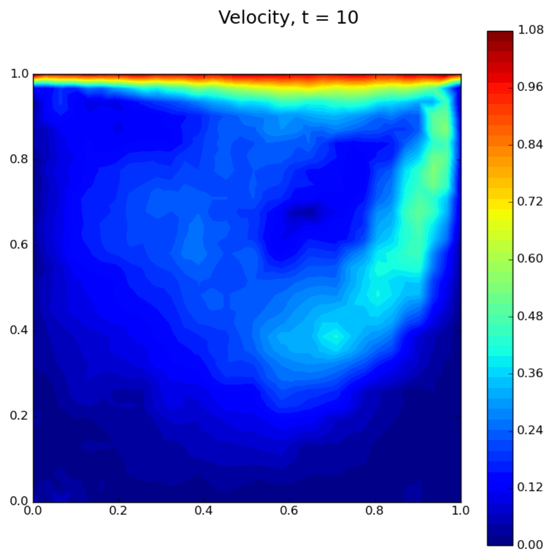

Figure C.131: Autoencoder velocity field for $N_H = 32 \times 32$ and Re = 500 for t = 10

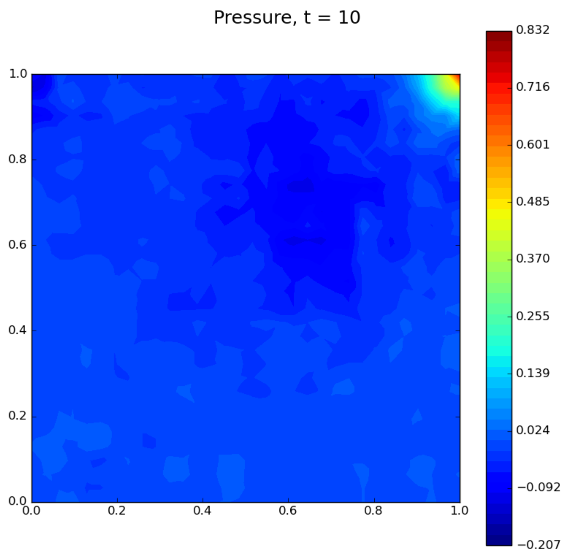

Figure C.132: Autoencoder pressure field for $N_H = 32 \times 32$ and Re = 500 for t = 10

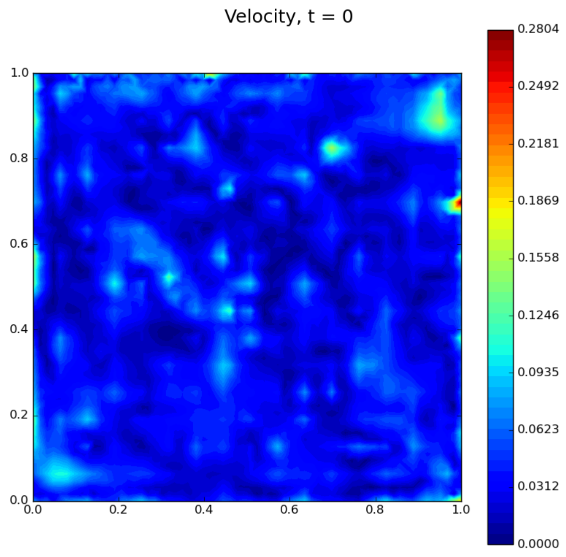

Figure C.133: Autoencoder velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 0

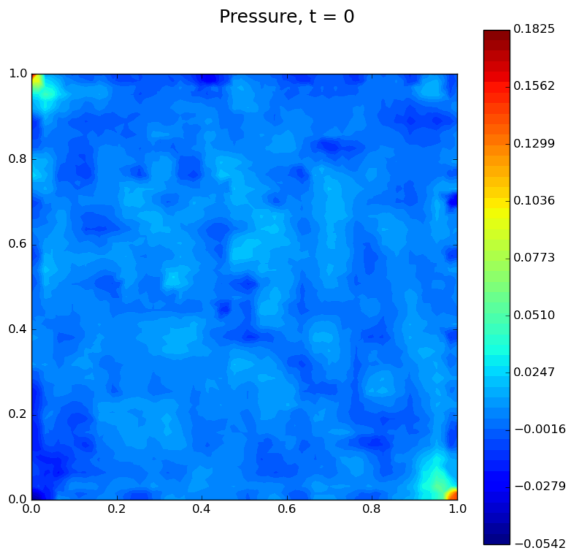

Figure C.134: Autoencoder pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 0

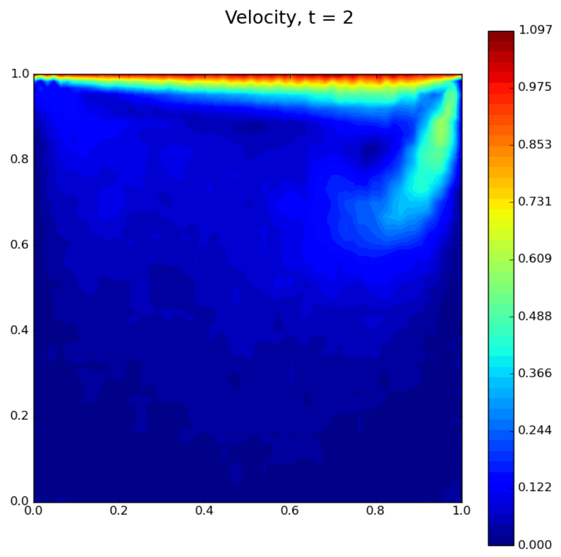

Figure C.135: Autoencoder velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 2

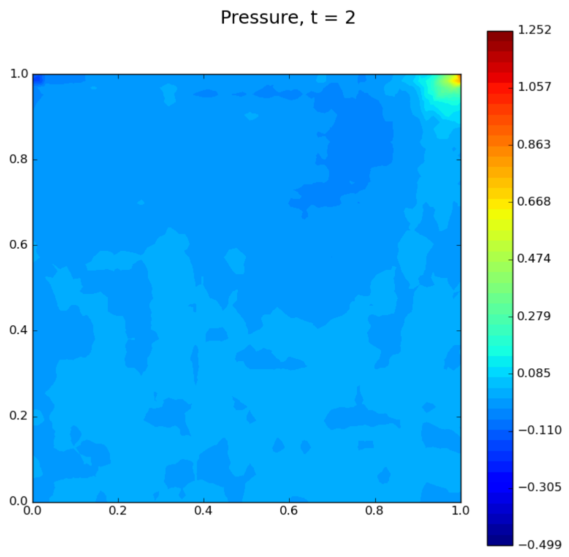

Figure C.136: Autoencoder pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 2

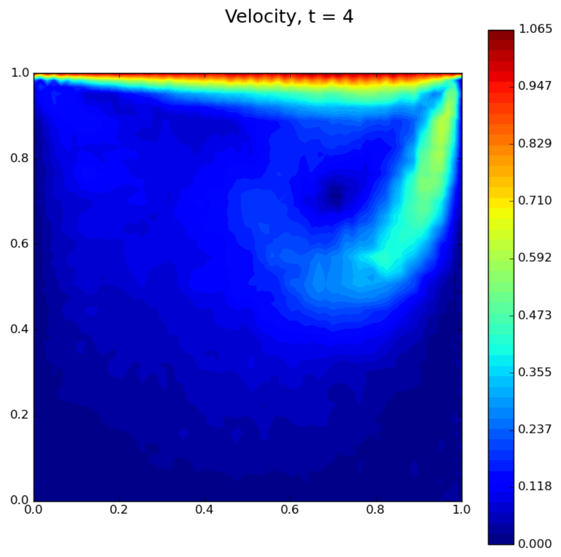

Figure C.137: Autoencoder velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 4

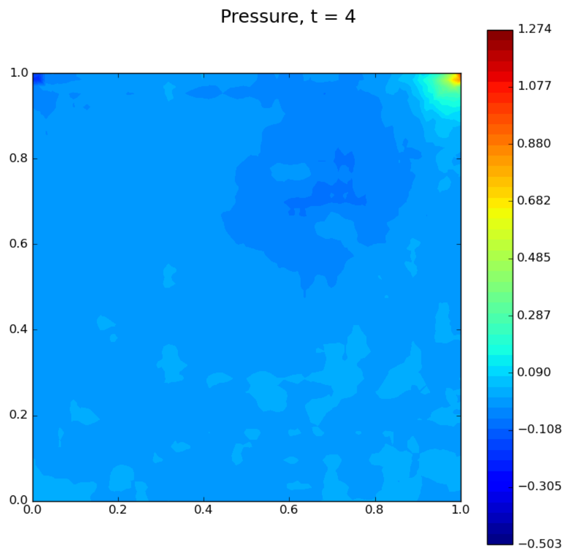

Figure C.138: Autoencoder pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 4

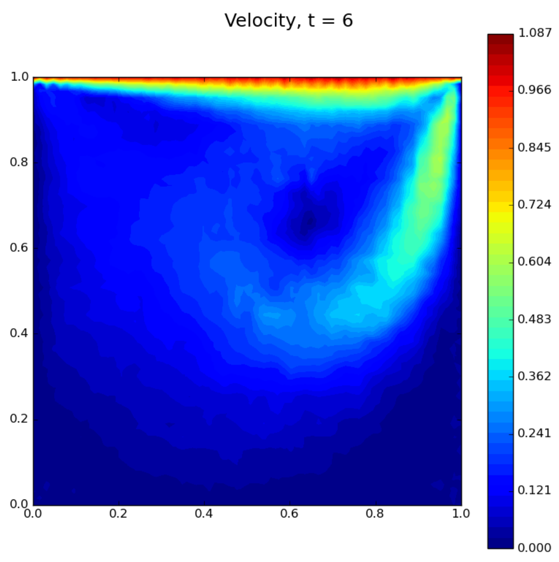

Figure C.139: Autoencoder velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 6

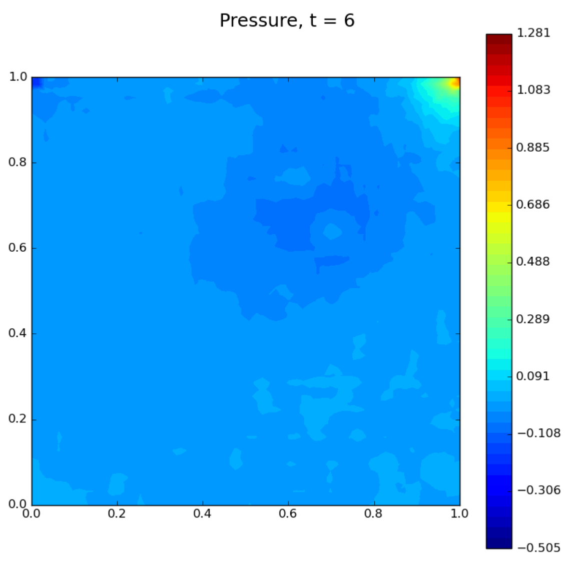

Figure C.140: Autoencoder pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 6

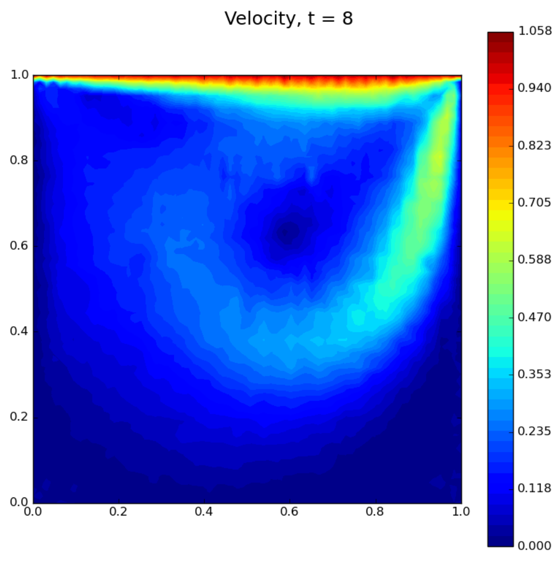

Figure C.141: Autoencoder velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 8

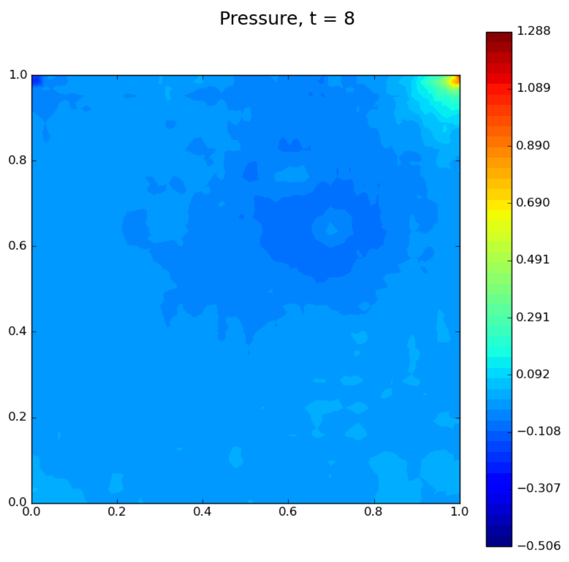

Figure C.142: Autoencoder pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 8

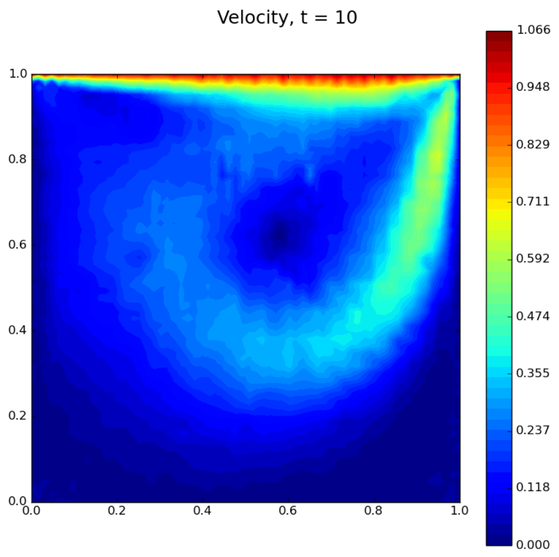

Figure C.143: Autoencoder velocity field for $N_H = 64 \times 64$ and Re = 500 for t = 10

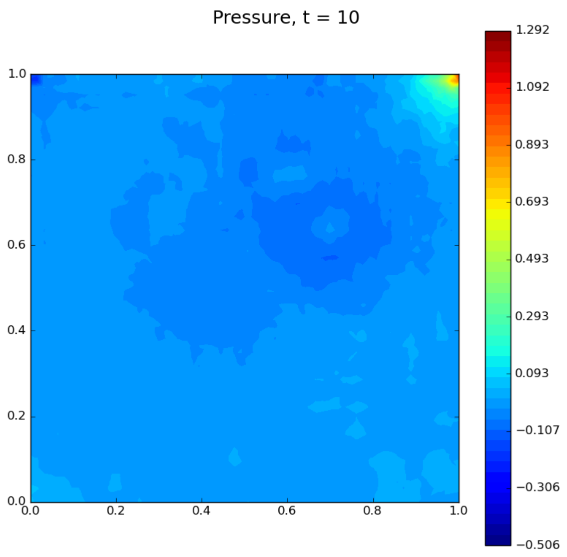

Figure C.144: Autoencoder pressure field for $N_H = 64 \times 64$ and Re = 500 for t = 10

C.2. Lid-driven Cavity Flow Primal Error

C.2.1. POD

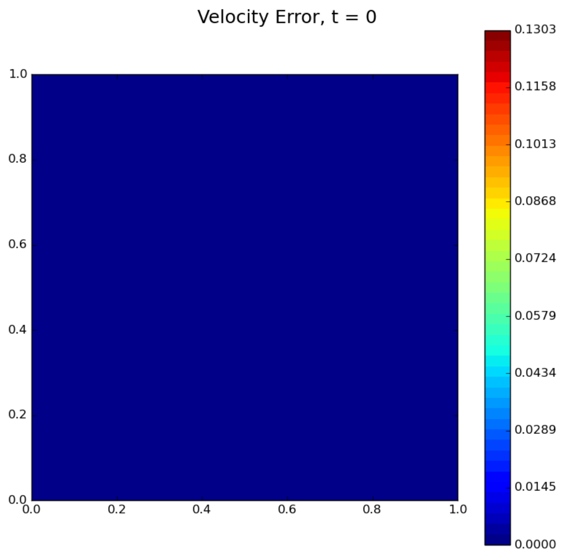

Figure C.145: POD velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 0

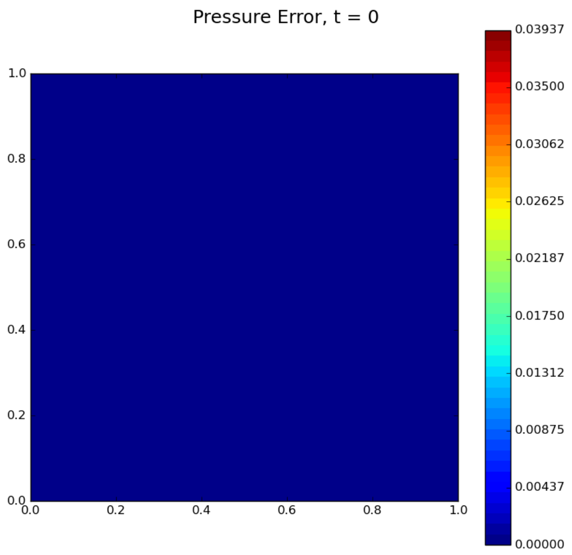

Figure C.146: POD pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 0

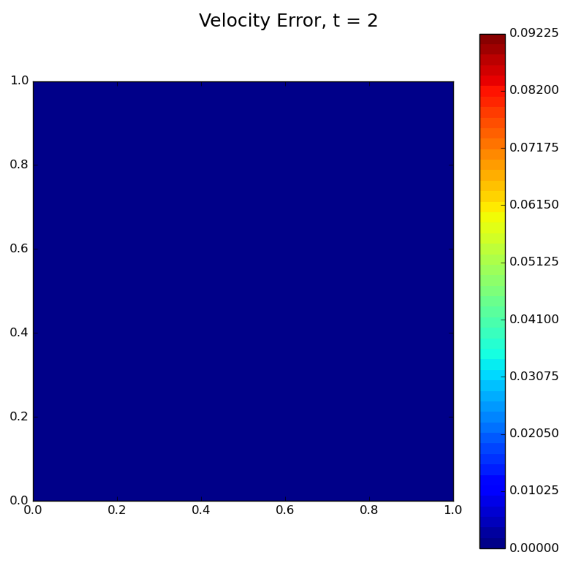

Figure C.147:
POD velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 2

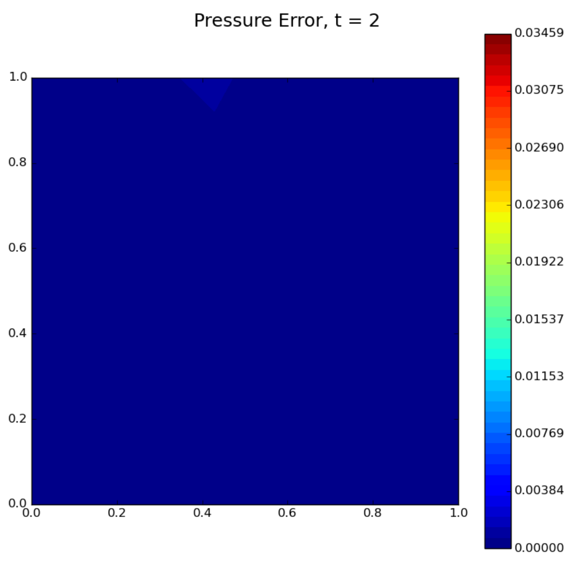

Figure C.148:
POD pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 2

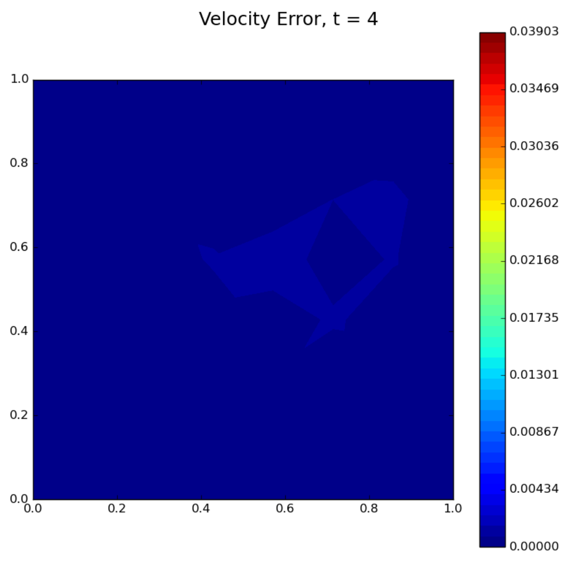

Figure C.149:
POD velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 4

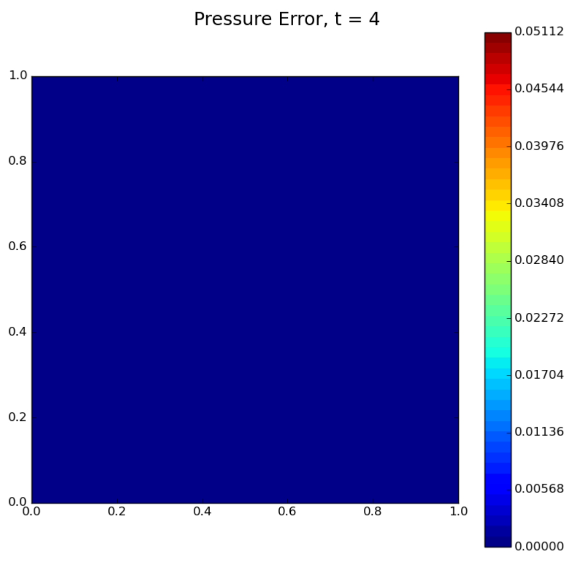

Figure C.150:
POD pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 4

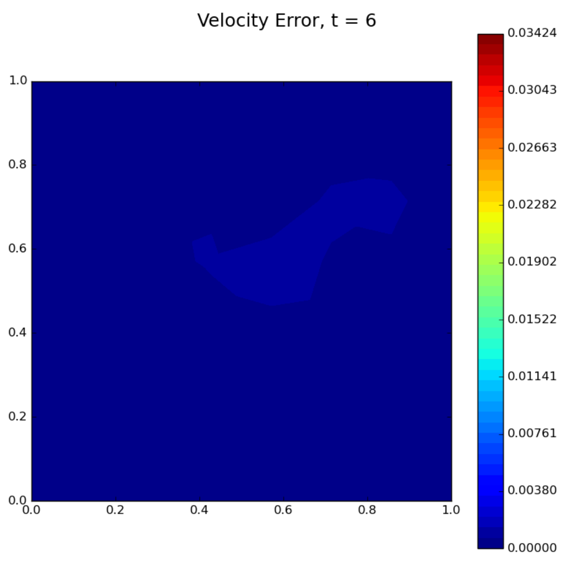

Figure C.151:
POD velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 6

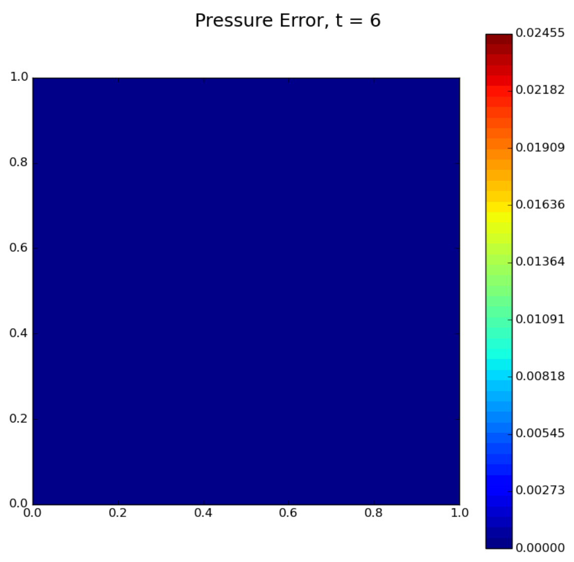

Figure C.152:
POD pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 6

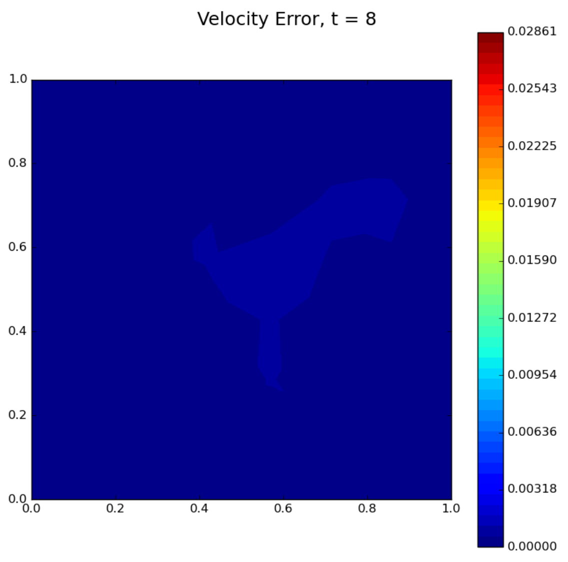

Figure C.153:
POD velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 8

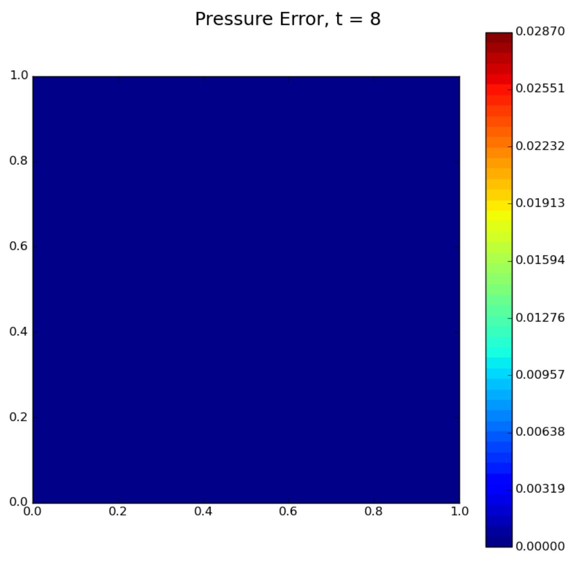

Figure C.154:
POD pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 8

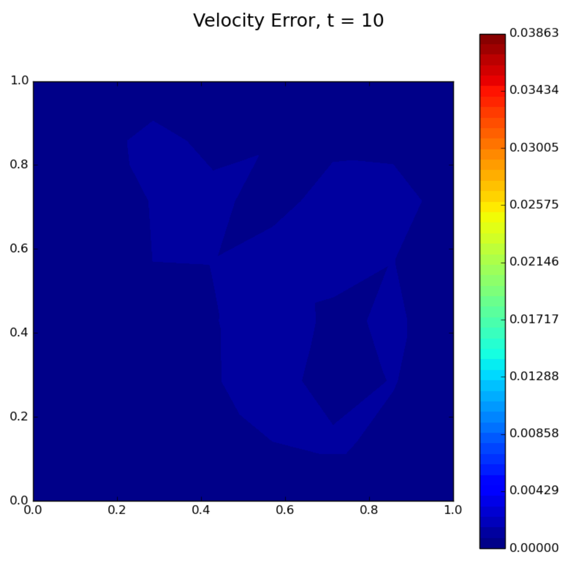

Figure C.155:
POD velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 10

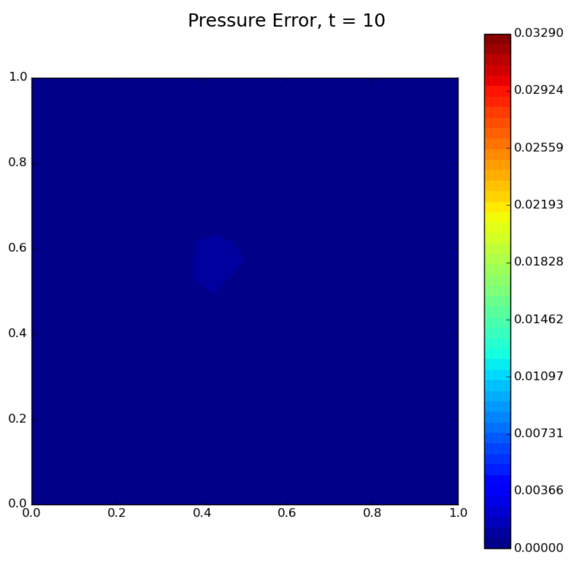

Figure C.156:
POD pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 10

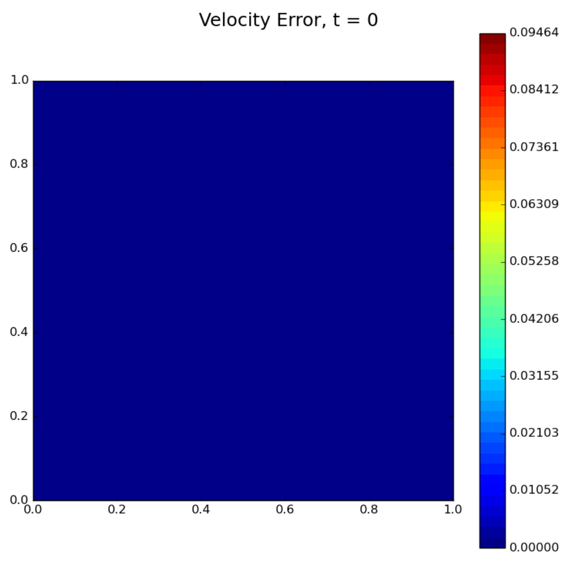

Figure C.157:
POD velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 0

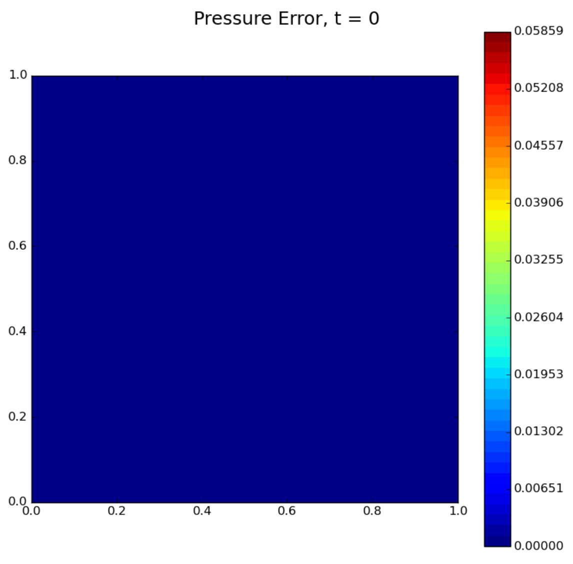

Figure C.158: POD
pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 0

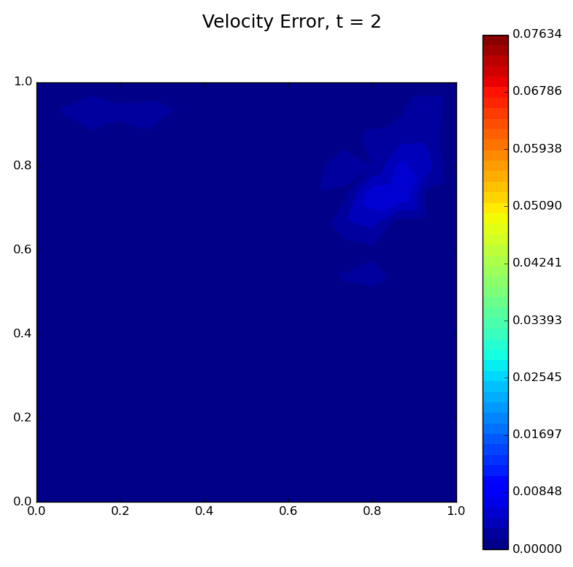

Figure C.159:
POD velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 2

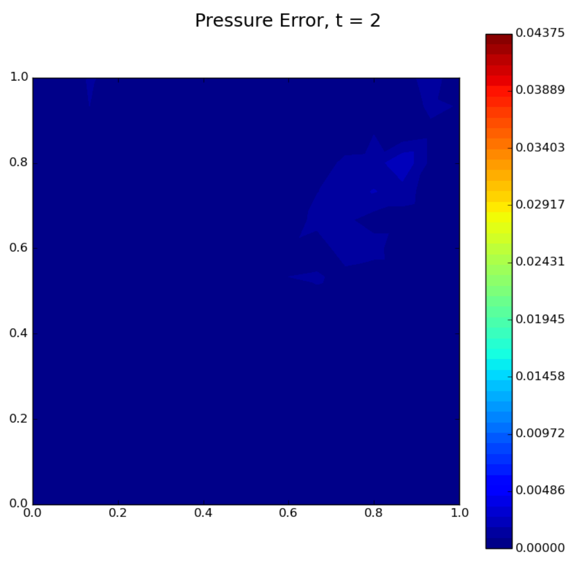

Figure C.160: POD
pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 2

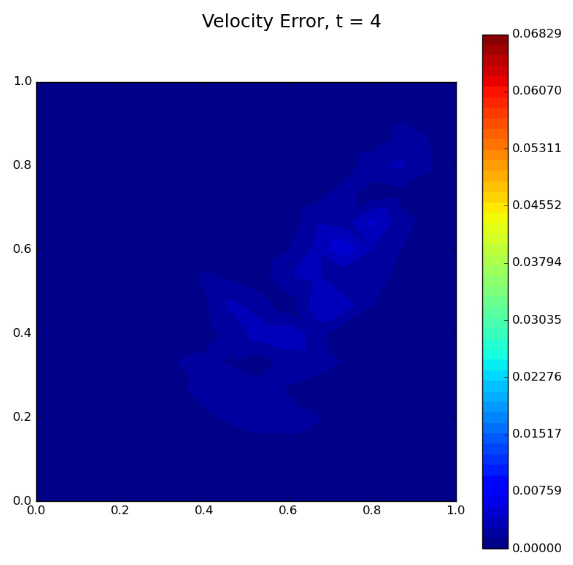

Figure C.161:
POD velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 4

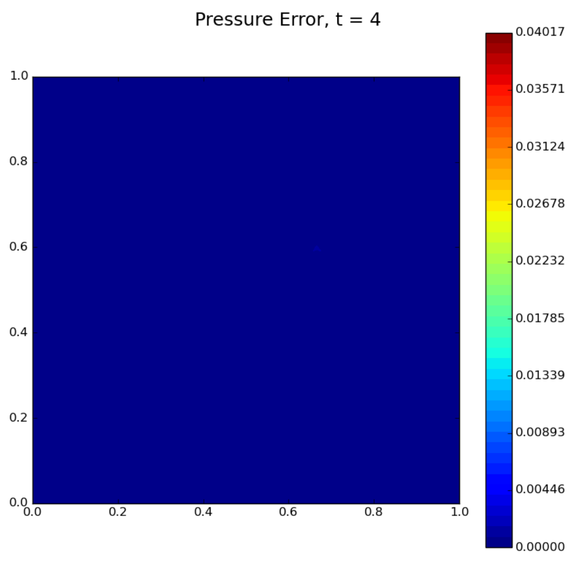

Figure C.162: POD
pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 4

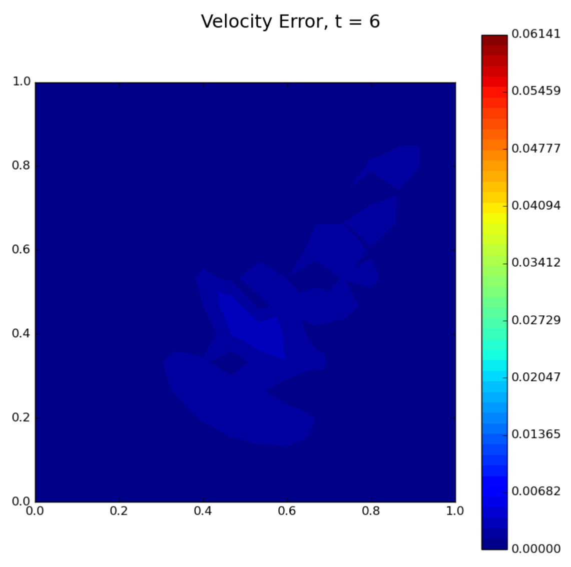

Figure C.163:
POD velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 6

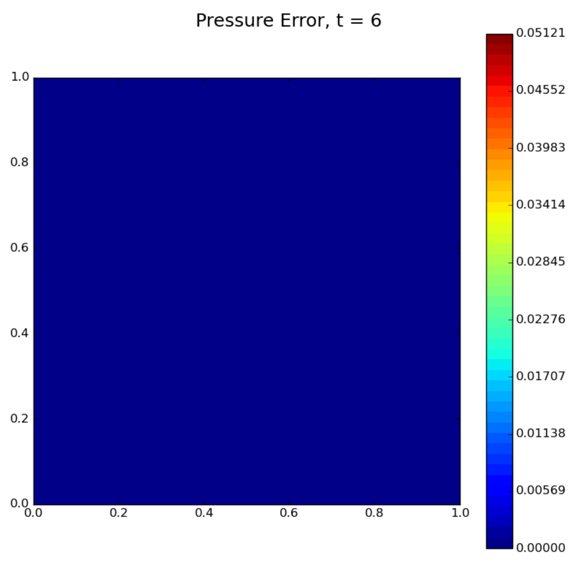

Figure C.164: POD
pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 6

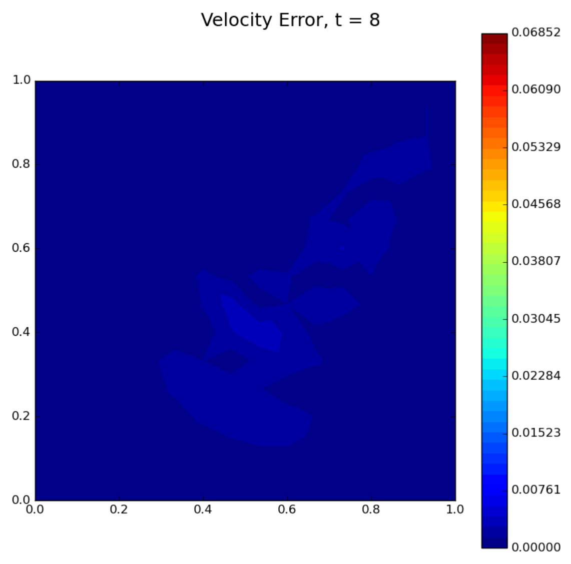

Figure C.165:
POD velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 8

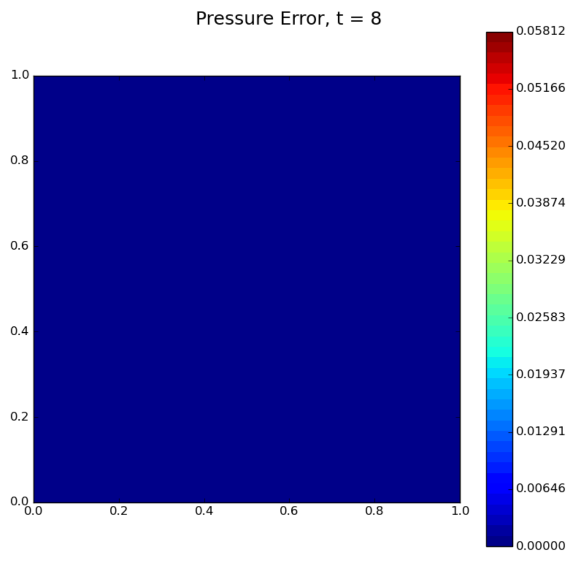

Figure C.166: POD
pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 8

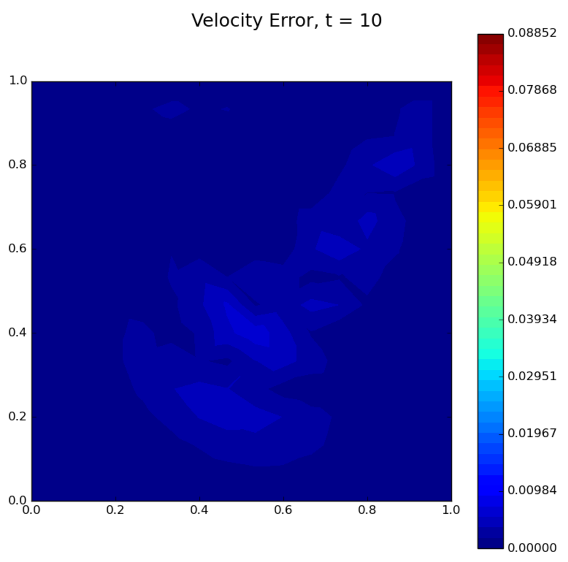

Figure C.167: POD velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 10

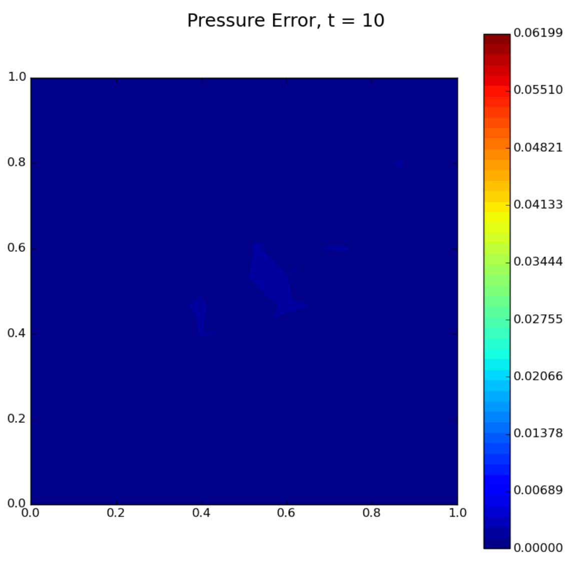

Figure C.168: POD pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 10

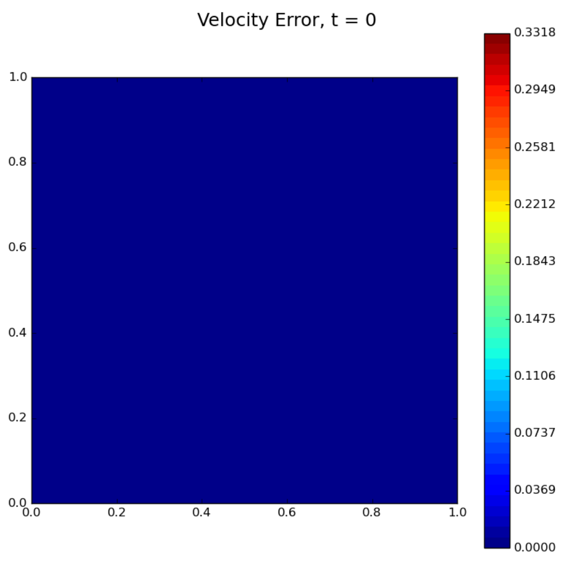

Figure C.169: POD velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 0

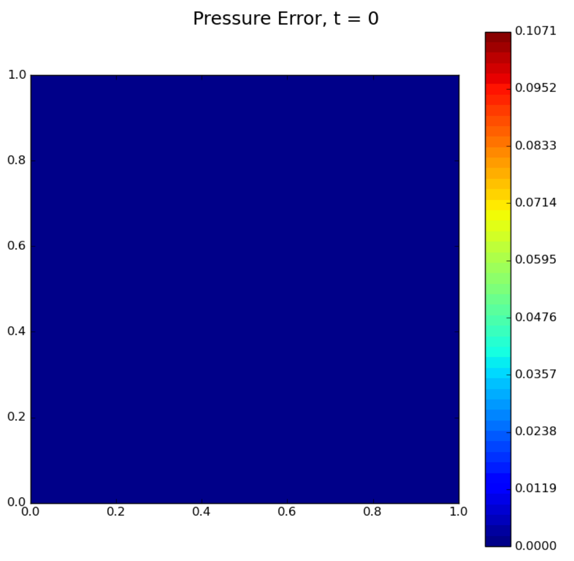

Figure C.170: POD pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 0

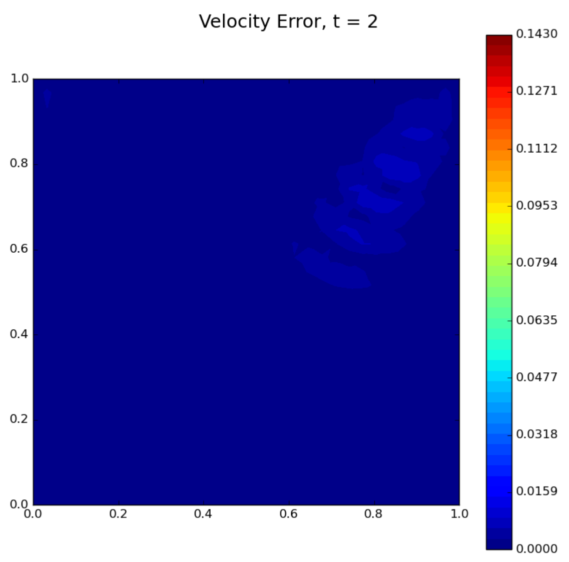

Figure C.171:
POD velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 2

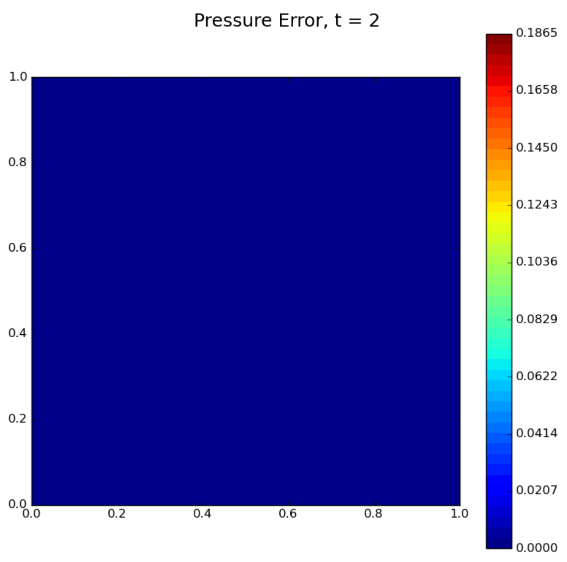

Figure C.172: POD
pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 2

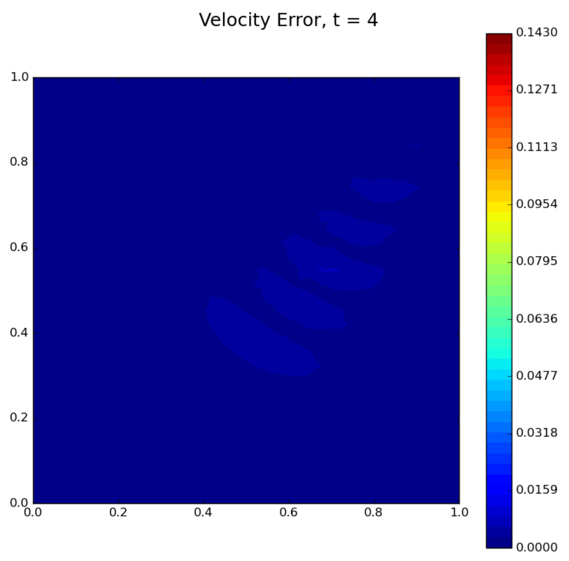

Figure C.173:
POD velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 4

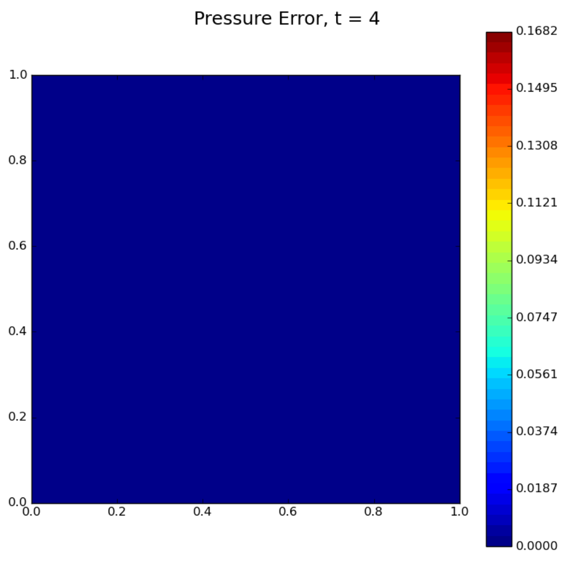

Figure C.174: POD
pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 4

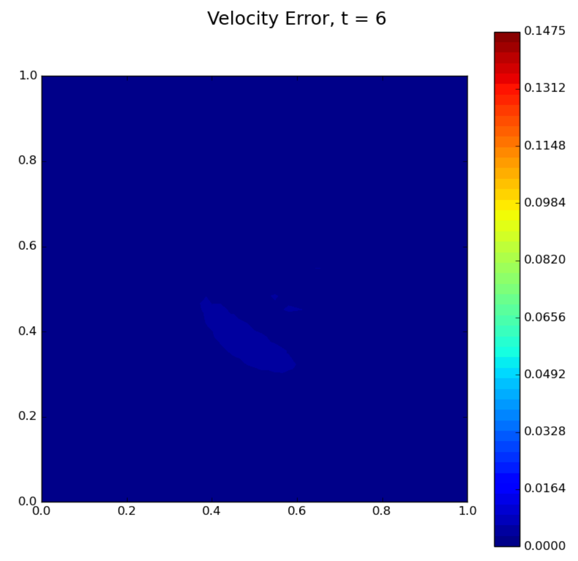

Figure C.175:
POD velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 6

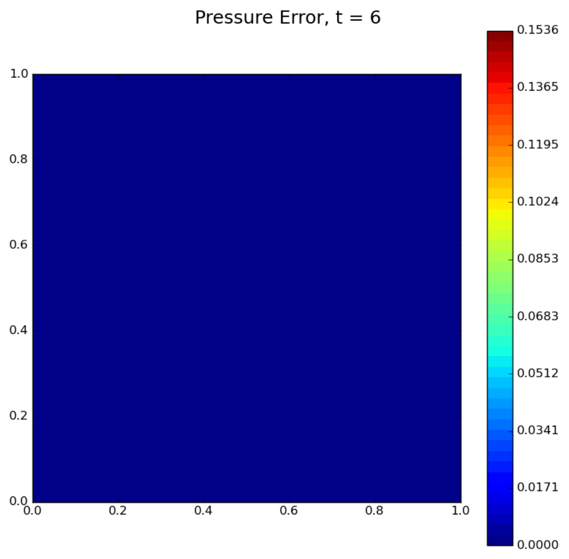

Figure C.176: POD
pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 6

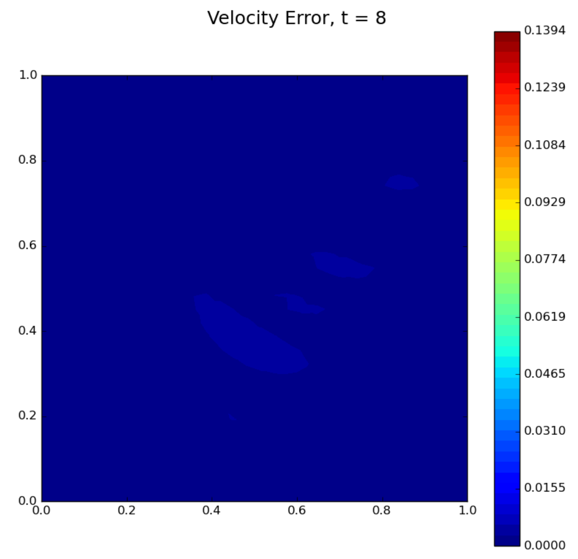

Figure C.177:
POD velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 8

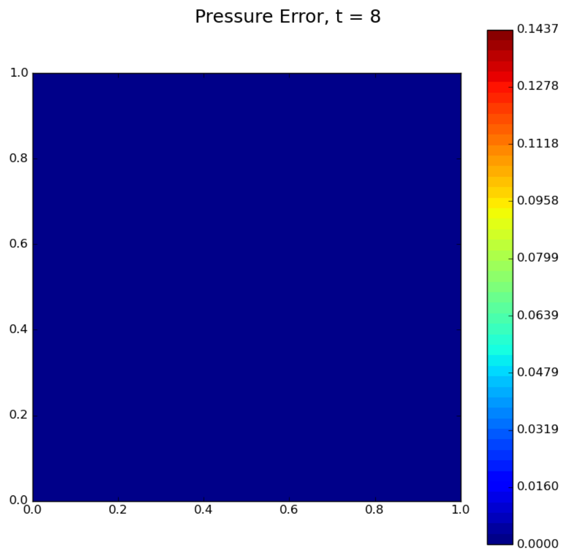

Figure C.178: POD
pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 8

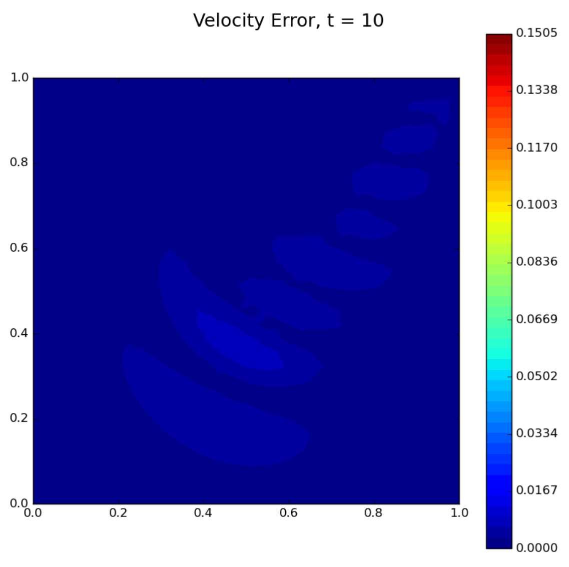

Figure C.179: POD velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 10

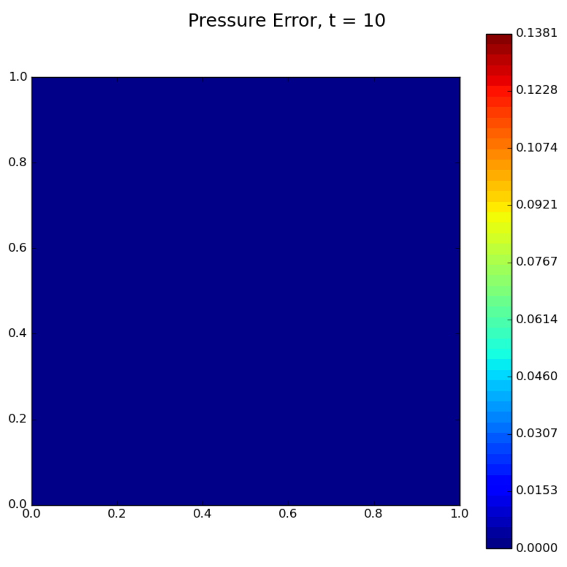

Figure C.180: POD pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 10

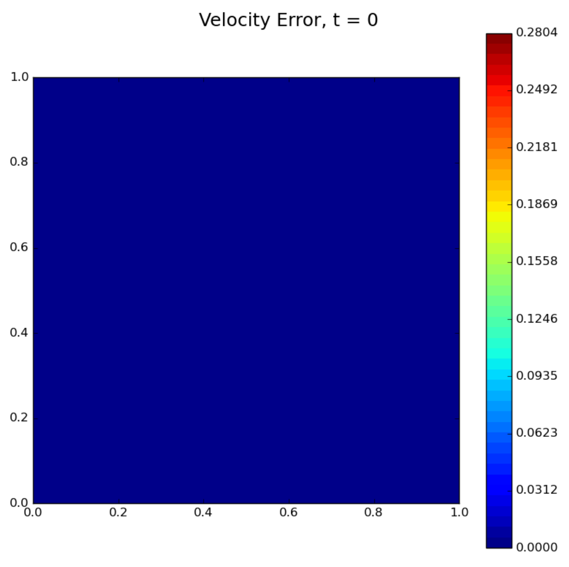

Figure C.181: POD velocity error field for $N_H = 64 \times 64$ and Re = 500 for t = 0

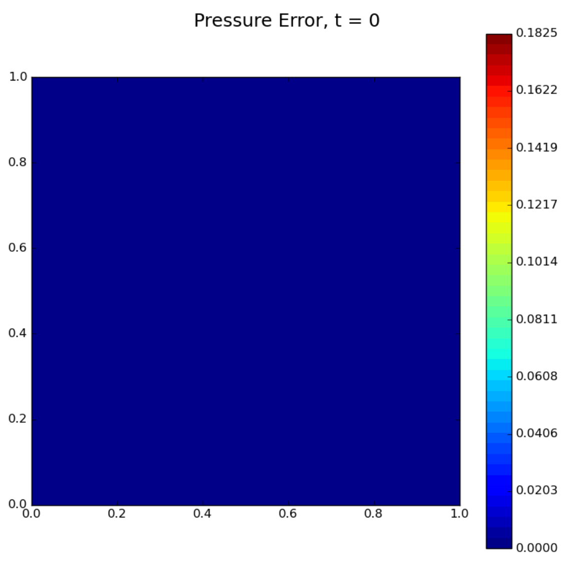

Figure C.182: POD pressure error field for $N_H = 64 \times 64$ and Re = 500 for t = 0

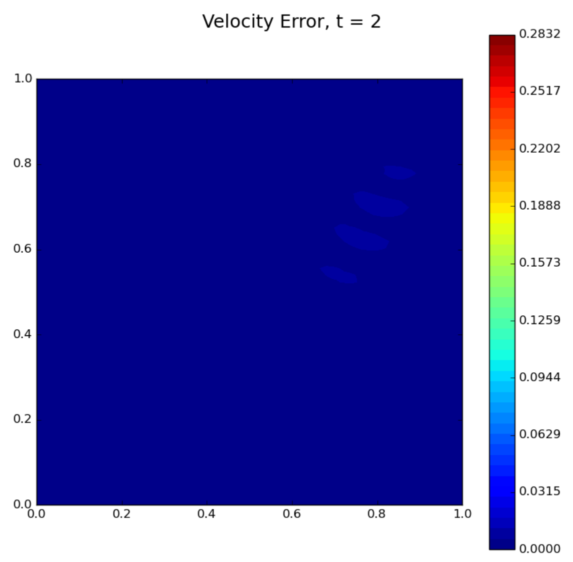

Figure C.183:
POD velocity error field for $N_H = 64\times64$ and Re = 500 for t = 2

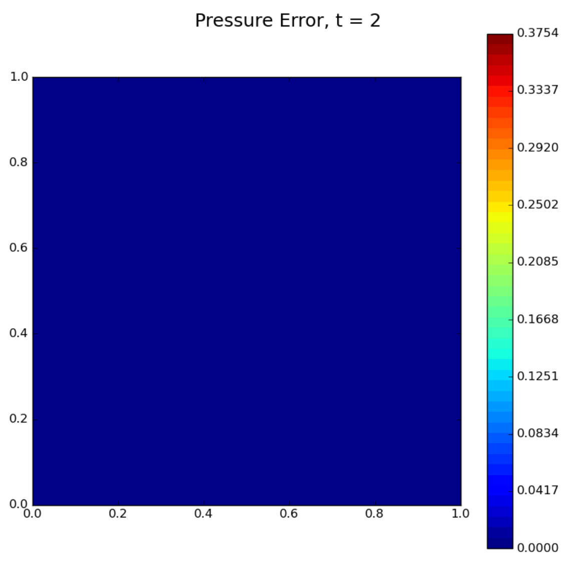

Figure C.184: POD
pressure error field for $N_H = 64\times64$ and Re = 500 for t = 2

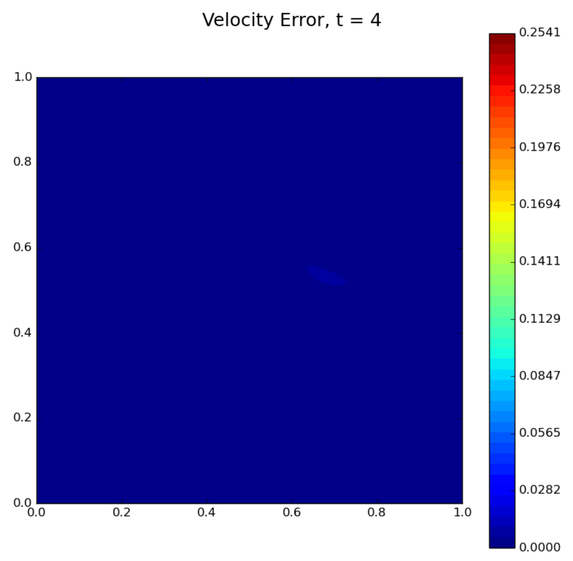

Figure C.185:
POD velocity error field for $N_H = 64\times64$ and Re = 500 for t = 4

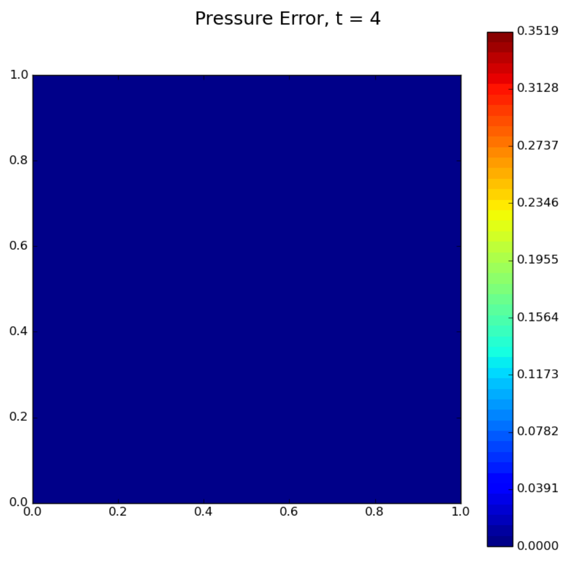

Figure C.186: POD
pressure error field for $N_H = 64\times64$ and Re = 500 for t = 4

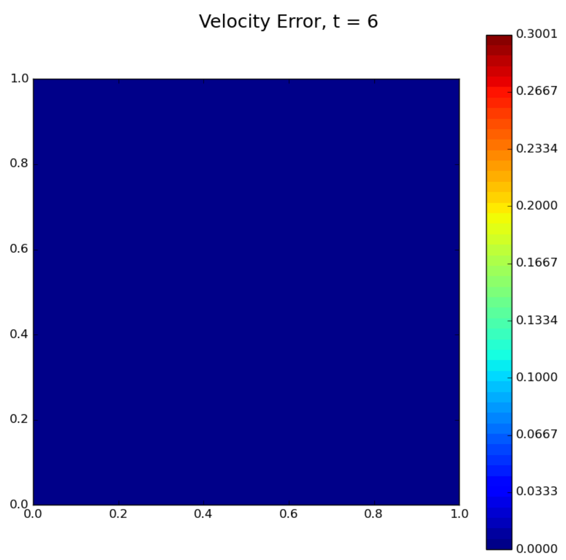

Figure C.187:
POD velocity error field for $N_H = 64 \times 64$ and Re = 500 for t = 6

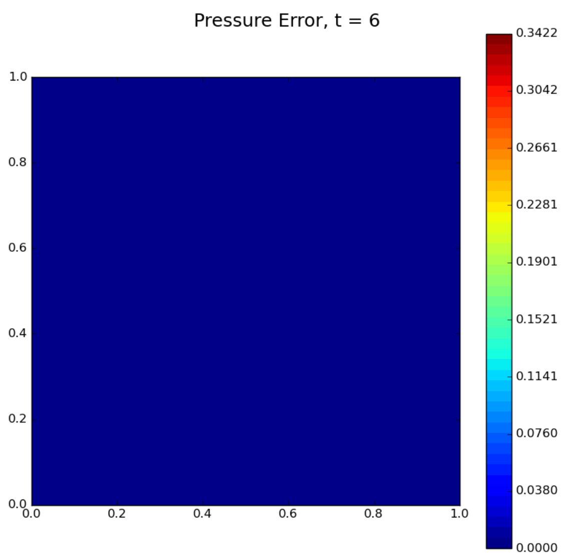

Figure C.188: POD pressure error field for $N_H = 64 \times 64$ and Re = 500 for t = 6

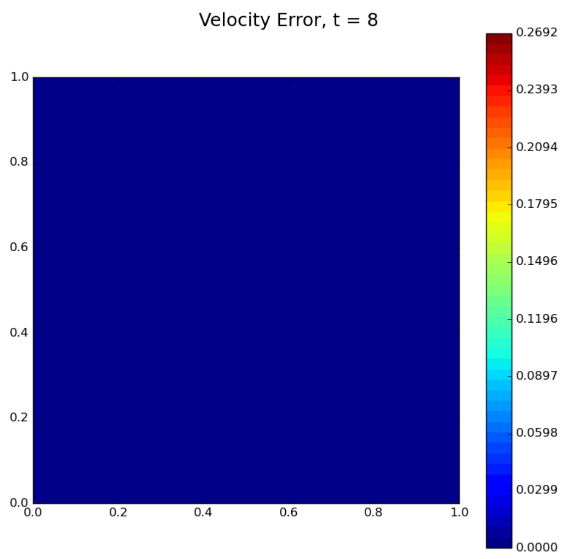

Figure C.189:
POD velocity error field for $N_H = 64 \times 64$ and Re = 500 for t = 8

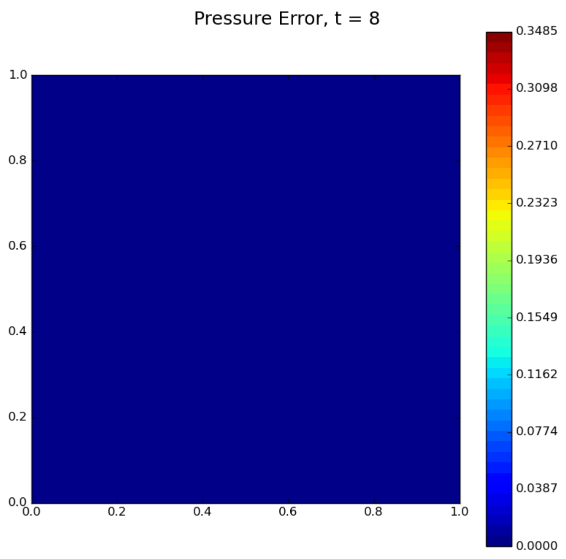

Figure C.190: POD pressure error field for $N_H = 64 \times 64$ and Re = 500 for t = 8

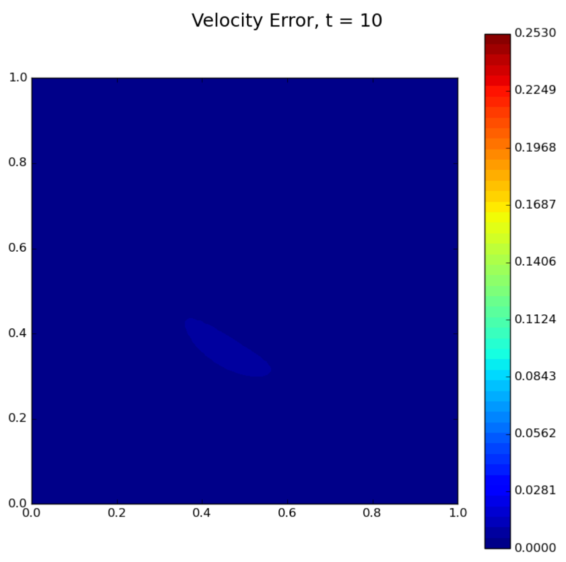

Figure C.191: POD
velocity error field for $N_H = 64 \times 64$ and Re = 500 for t = 10

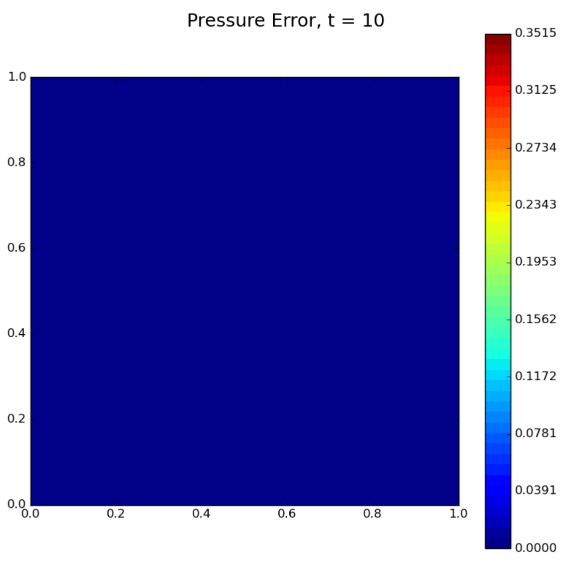

Figure C.192: POD
pressure error field for $N_H = 64 \times 64$ and Re = 500 for t = 10

C.2.2. Autoencoder

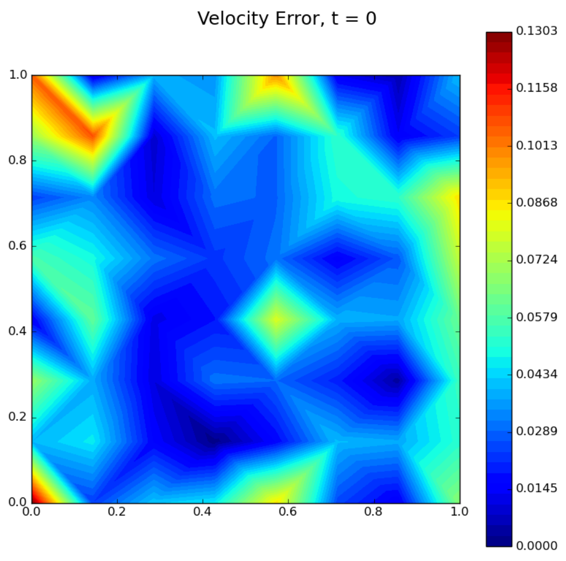

Figure C.193: Autoencoder
velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 0

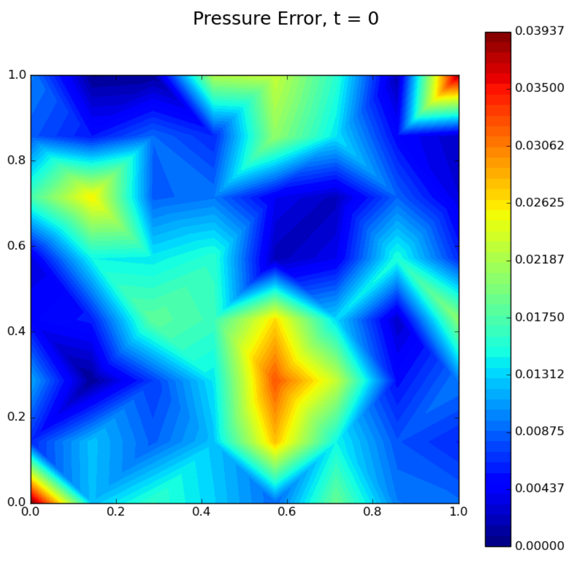

Figure C.194: Autoencoder
pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 0

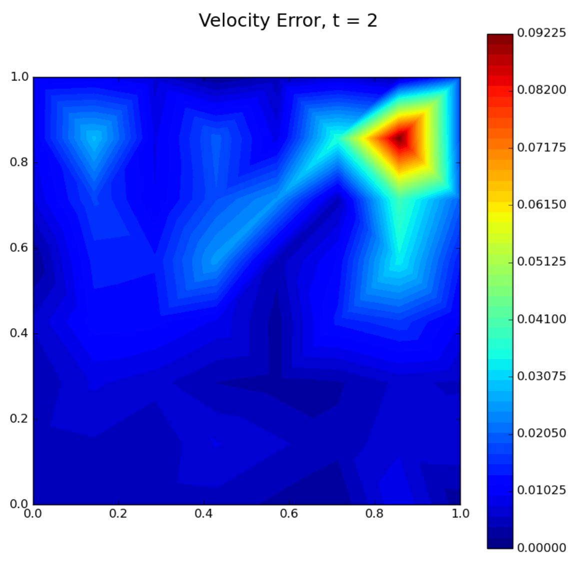

Figure C.195: Autoencoder velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 2

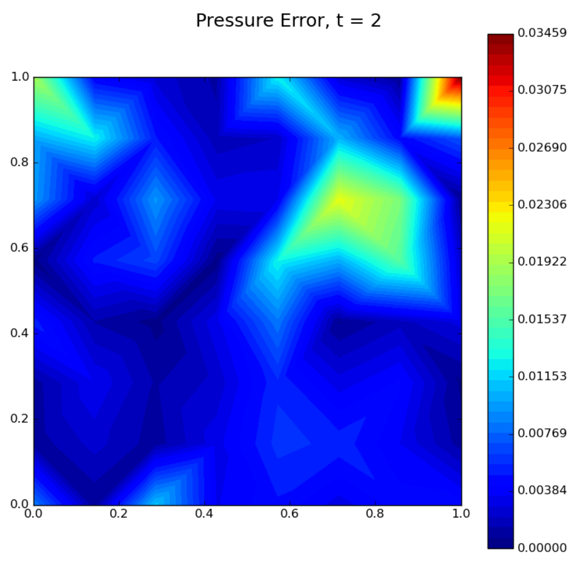

Figure C.196: Autoencoder pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 2

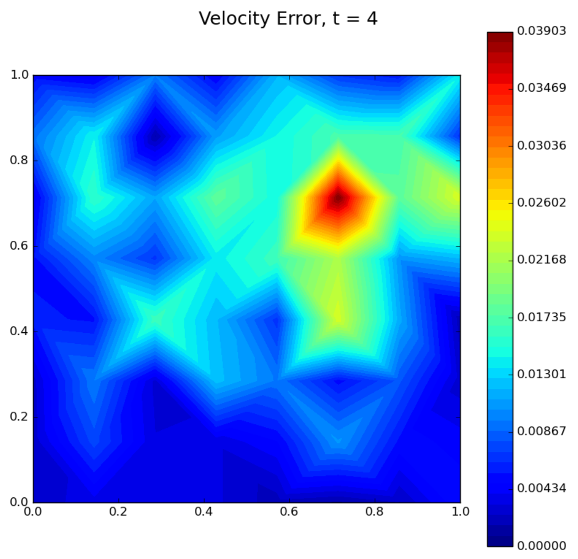

Figure C.197: Autoencoder velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 4

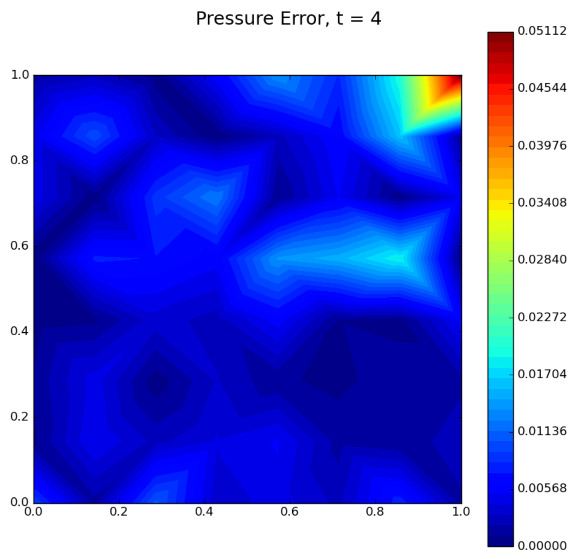

Figure C.198: Autoencoder pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 4

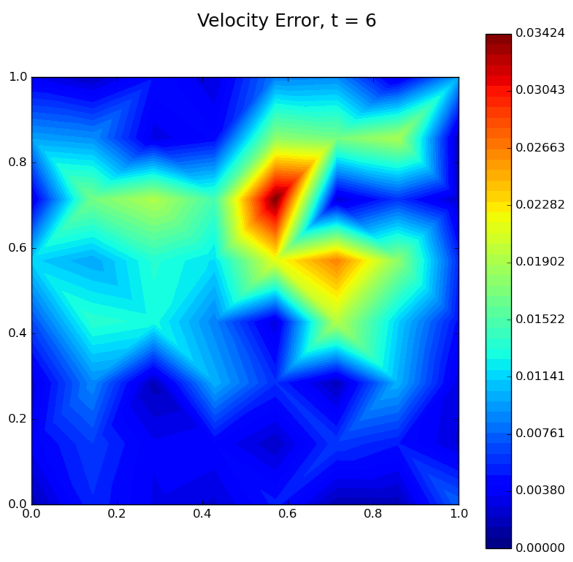

Figure C.199: Autoencoder velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 6

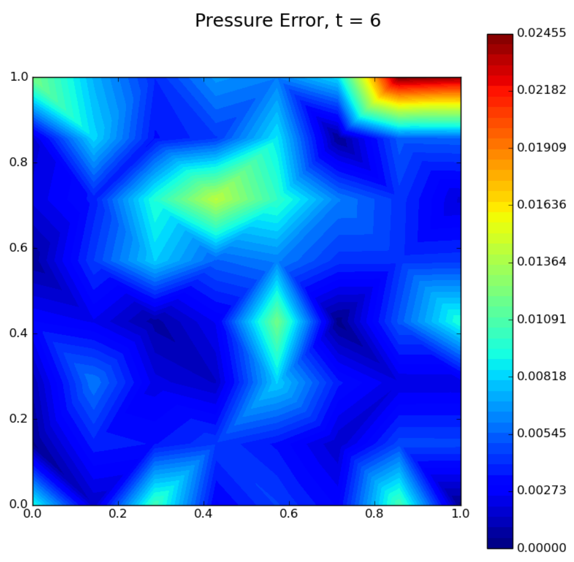

Figure C.200: Autoencoder pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 6

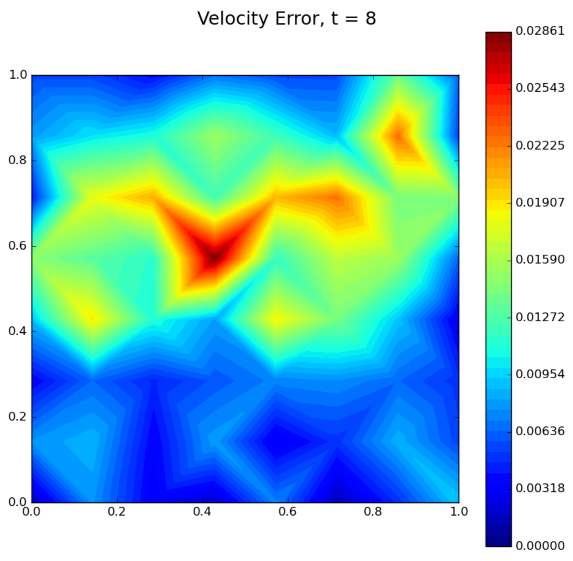

Figure C.201: Autoencoder velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 8

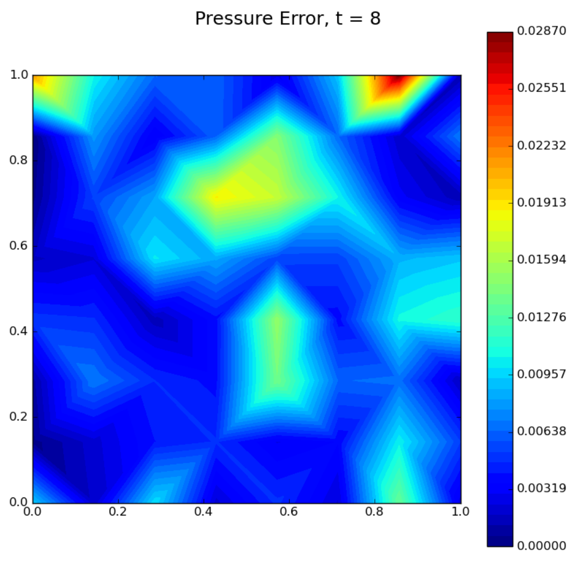

Figure C.202: Autoencoder pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 8

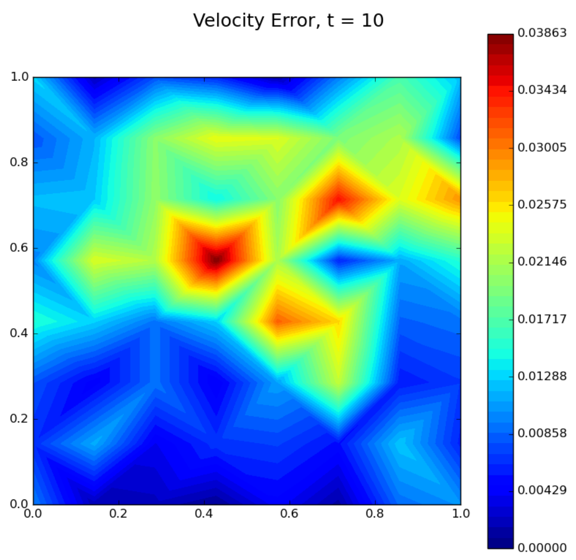

Figure C.203: Autoencoder velocity error field for $N_H = 8 \times 8$ and Re = 500 for t = 10

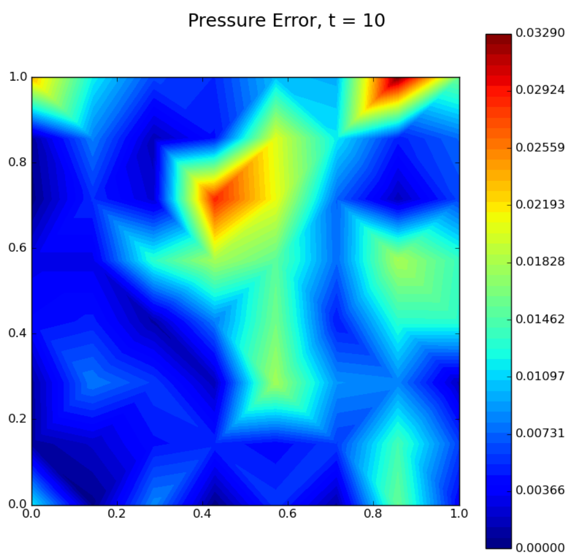

Figure C.204: Autoencoder pressure error field for $N_H = 8 \times 8$ and Re = 500 for t = 10

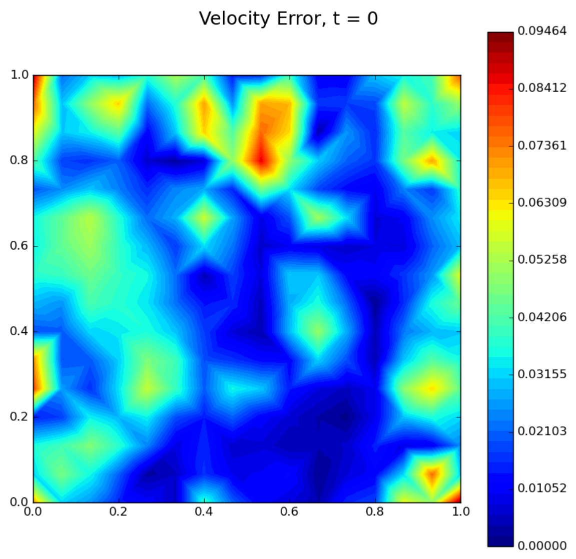

Figure C.205: Autoencoder velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 0

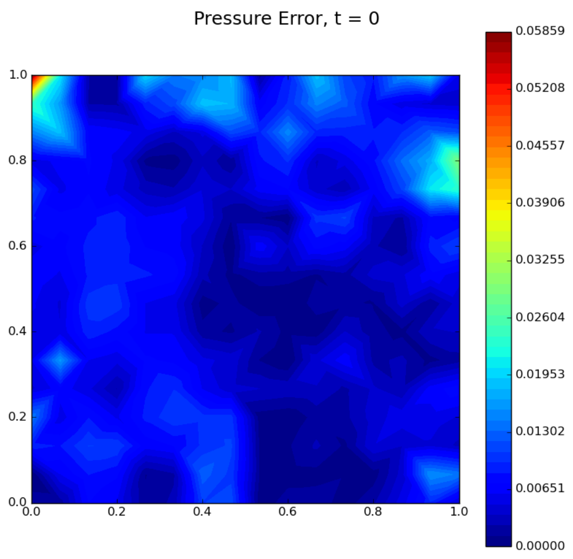

Figure C.206: Autoencoder pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 0

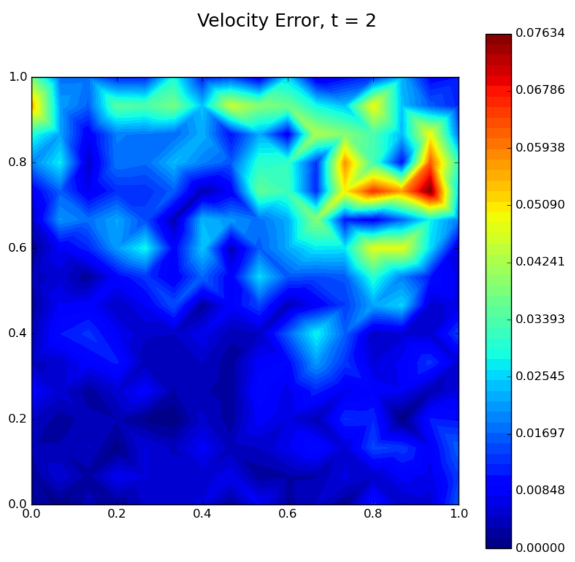

Figure C.207: Autoencoder
velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 2

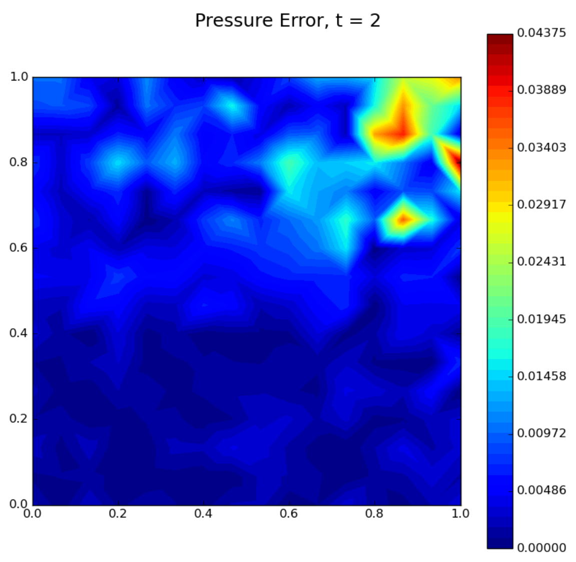

Figure C.208: Autoencoder
pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 2

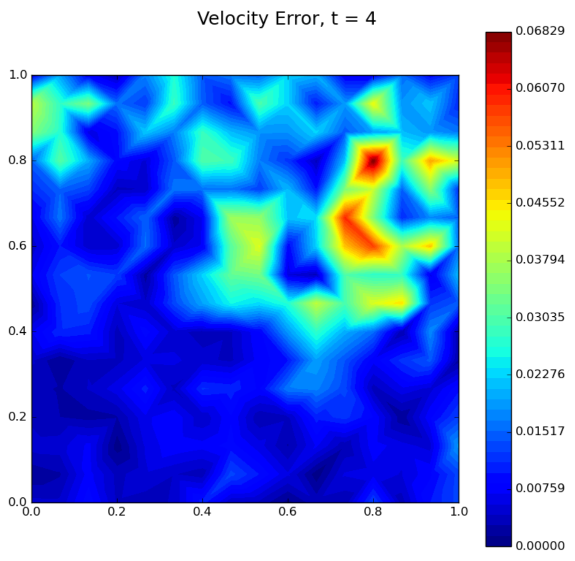

Figure C.209: Autoencoder
velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 4

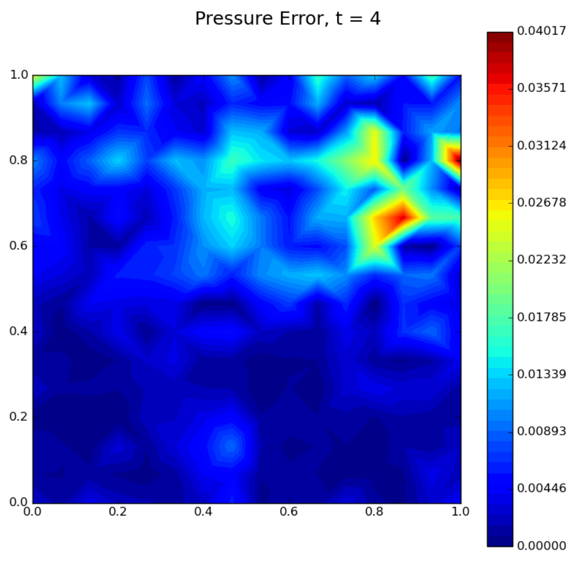

Figure C.210: Autoencoder
pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 4

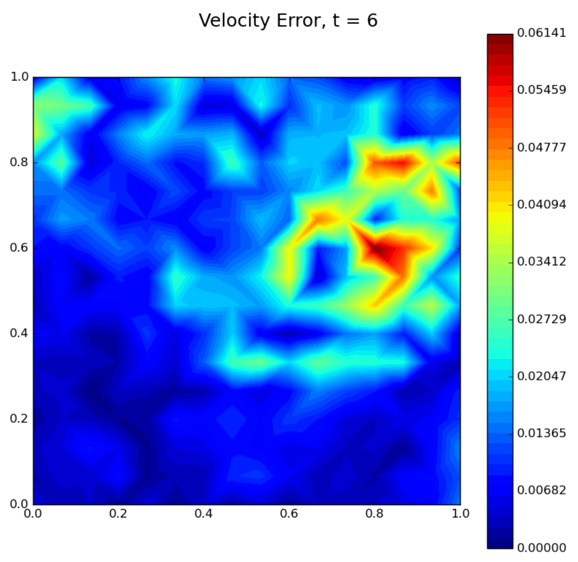

Figure C.211: Autoencoder velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 6

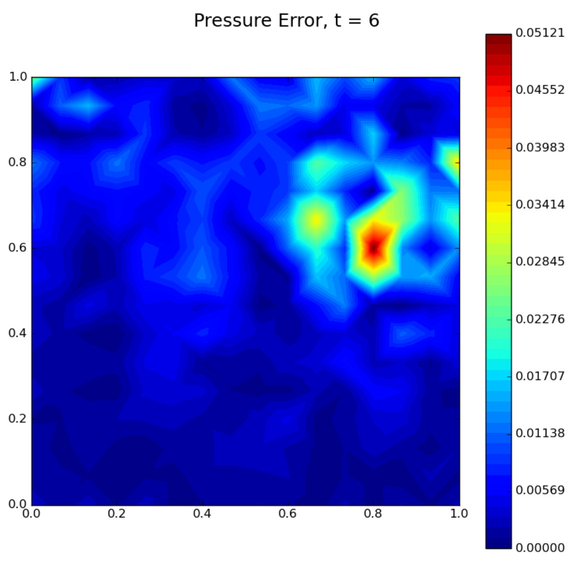

Figure C.212: Autoencoder pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 6

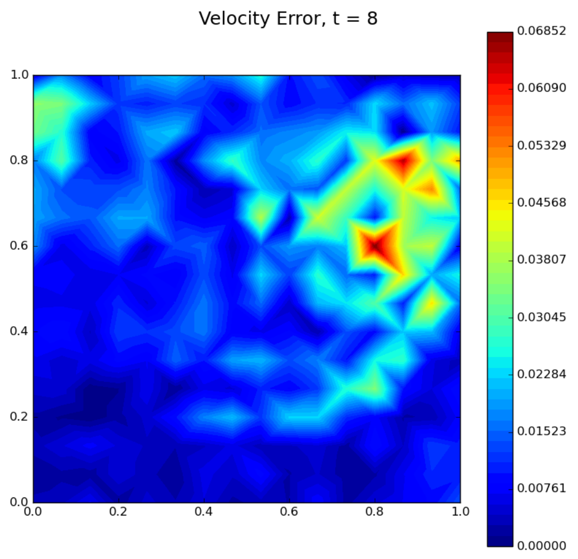

Figure C.213: Autoencoder velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 8

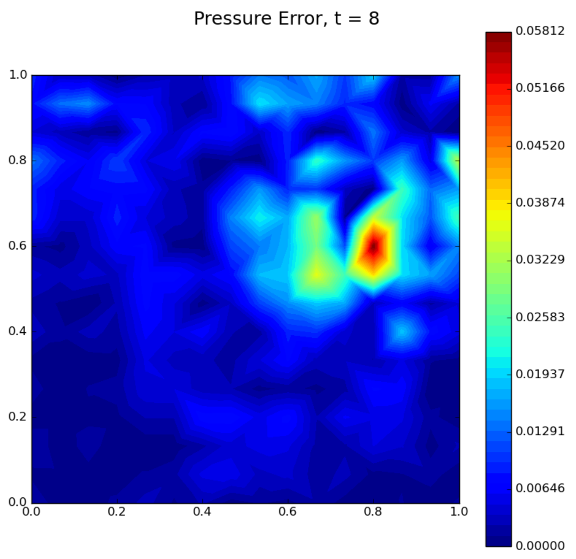

Figure C.214: Autoencoder pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 8

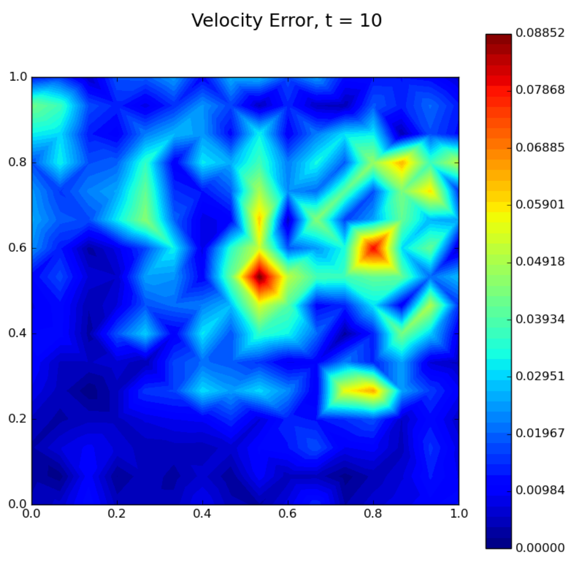

Figure C.215: Autoencoder velocity error field for $N_H = 16 \times 16$ and Re = 500 for t = 10

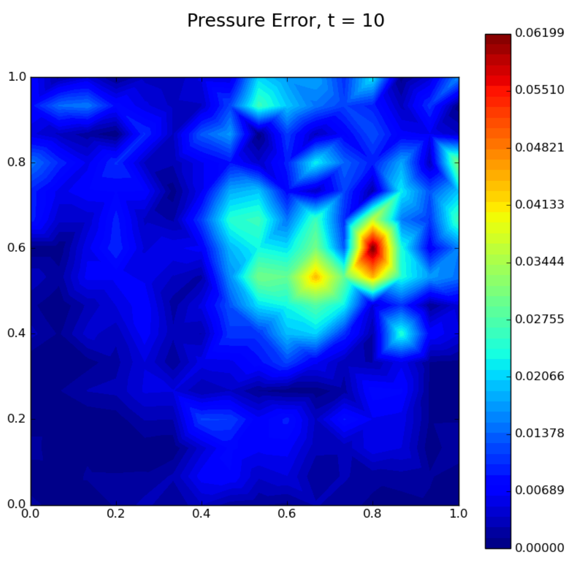

Figure C.216: Autoencoder pressure error field for $N_H = 16 \times 16$ and Re = 500 for t = 10

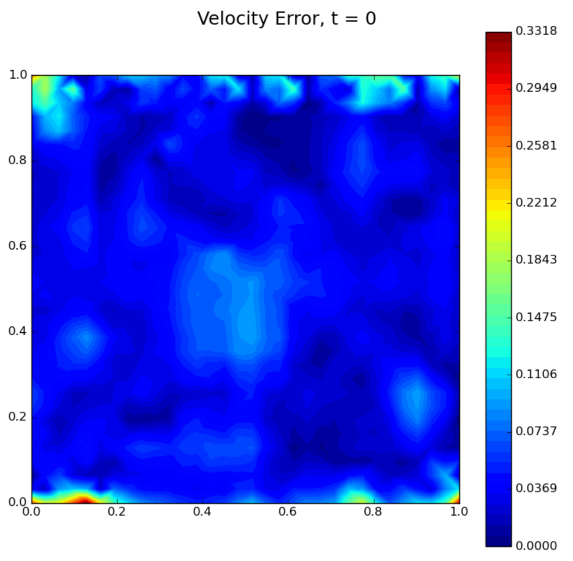

Figure C.217: Autoencoder velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 0

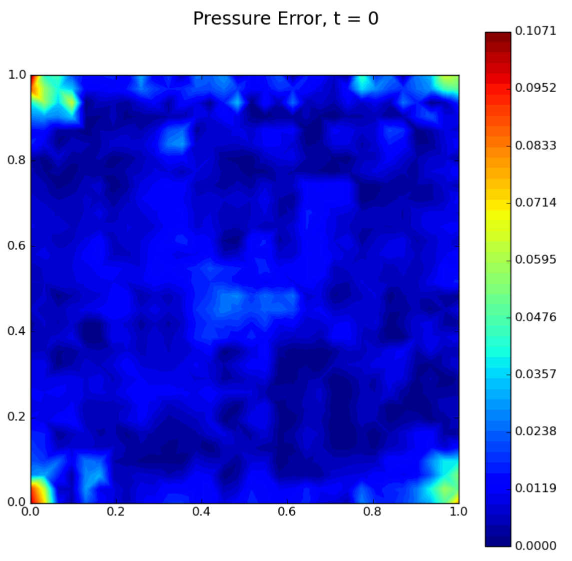

Figure C.218: Autoencoder pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 0

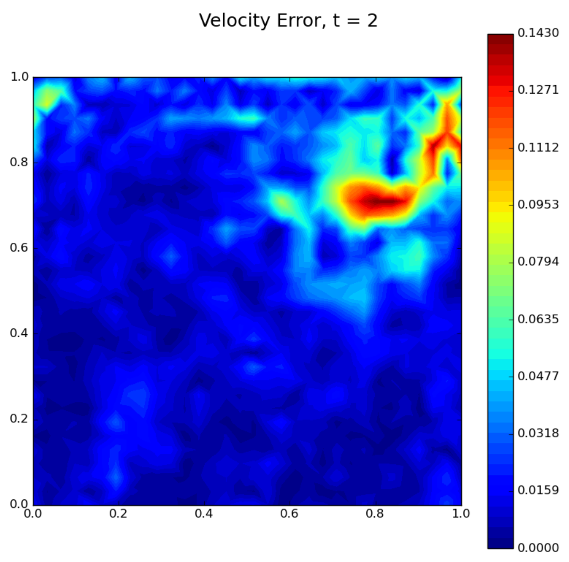

Figure C.219: Autoencoder
velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 2

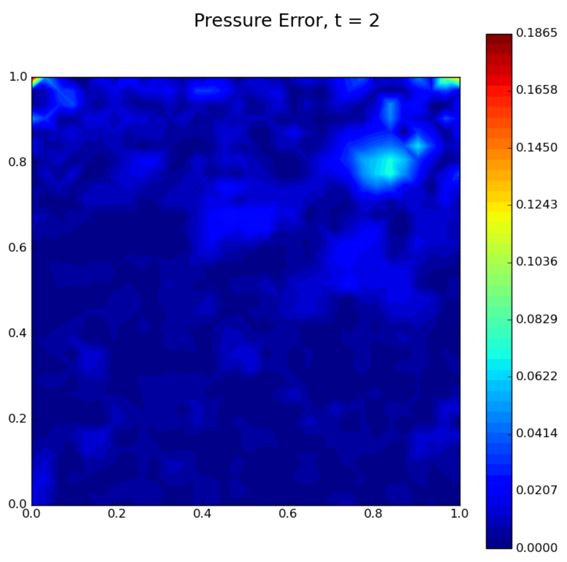

Figure C.220: Autoencoder
pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 2

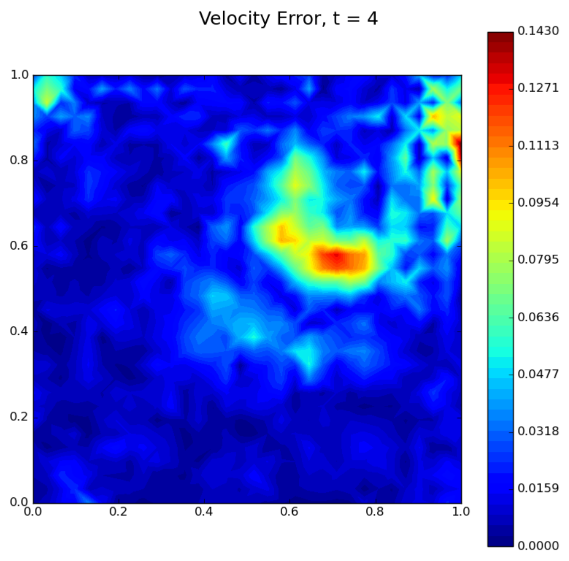

Figure C.221: Autoencoder
velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 4

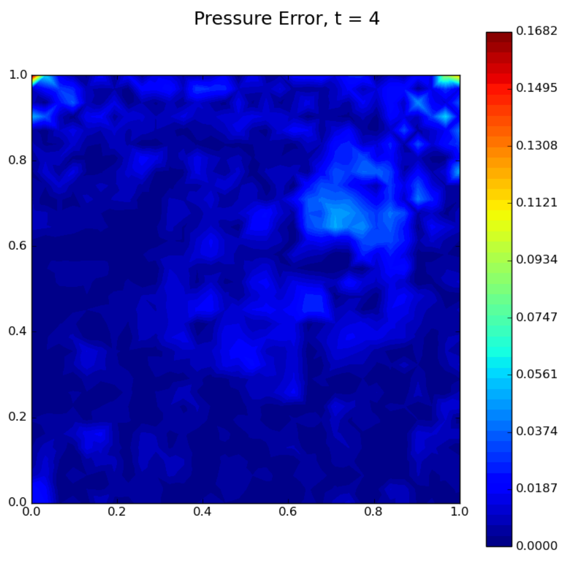

Figure C.222: Autoencoder
pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 4

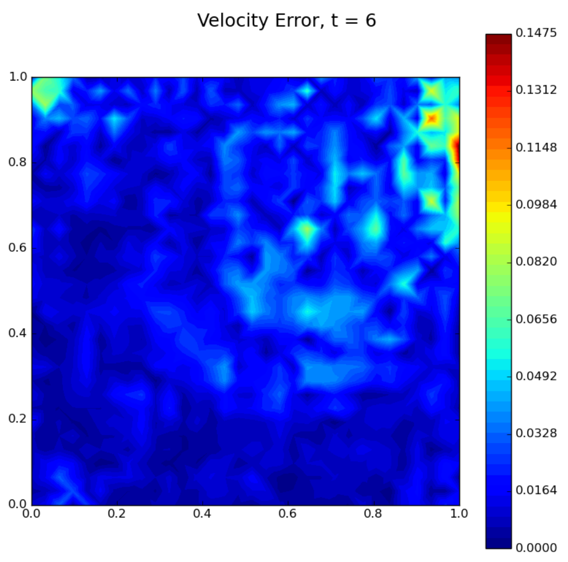

Figure C.223: Autoencoder velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 6

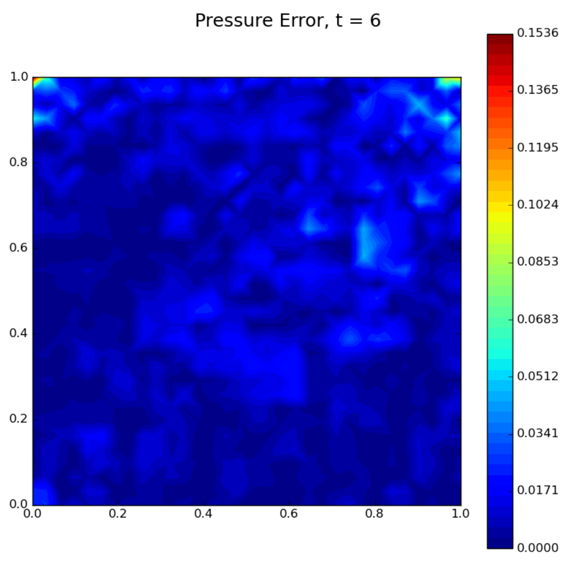

Figure C.224: Autoencoder pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 6

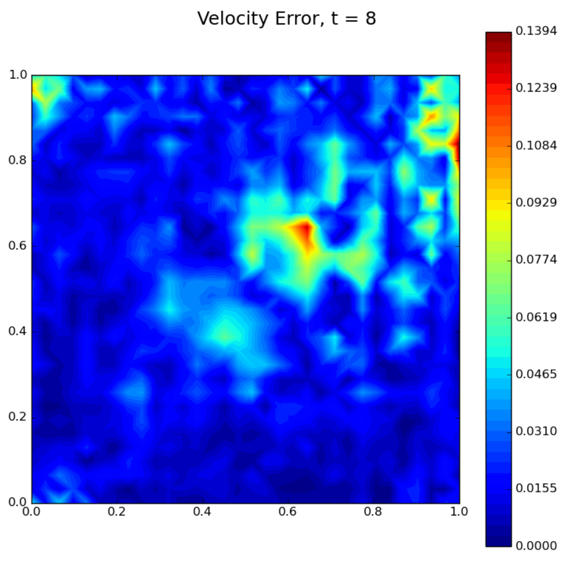

Figure C.225: Autoencoder velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 8

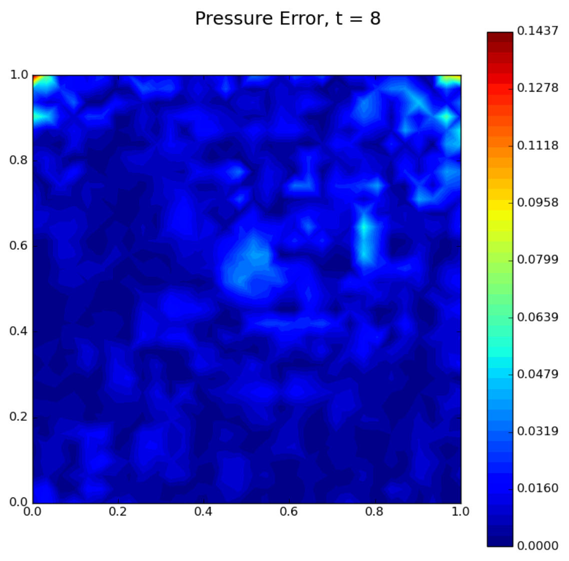

Figure C.226: Autoencoder pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 8

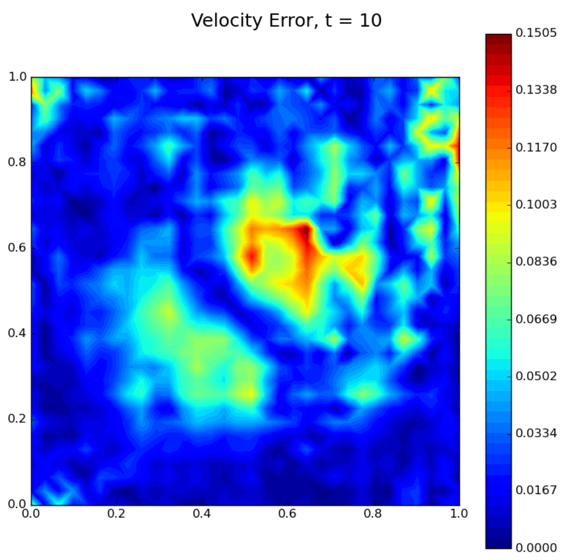

Figure C.227: Autoencoder velocity error field for $N_H = 32 \times 32$ and Re = 500 for t = 10

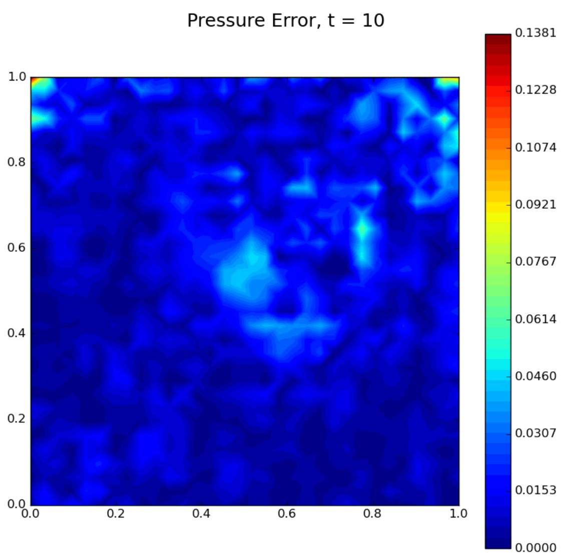

Figure C.228: Autoencoder pressure error field for $N_H = 32 \times 32$ and Re = 500 for t = 10

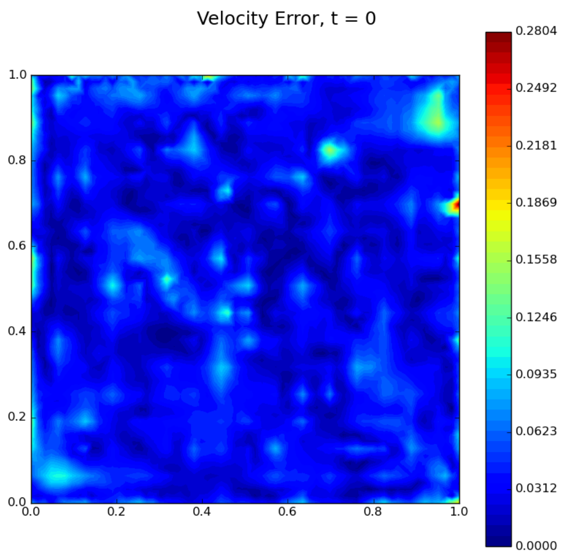

Figure C.229: Autoencoder velocity error field for $N_H = 64 \times 64$ and Re = 500 for t = 0

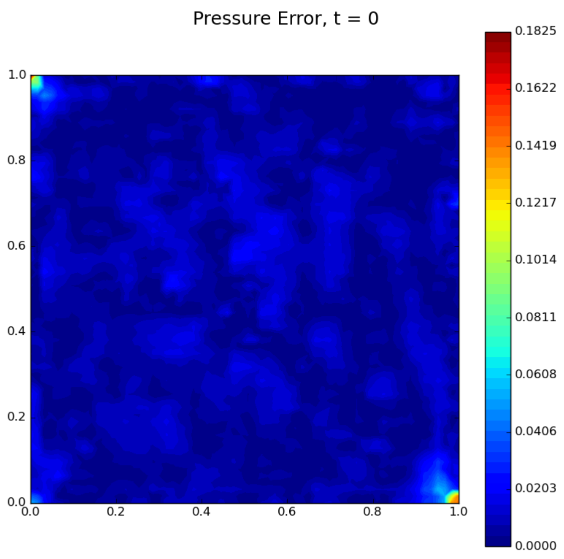

Figure C.230: Autoencoder pressure error field for $N_H = 64 \times 64$ and Re = 500 for t = 0

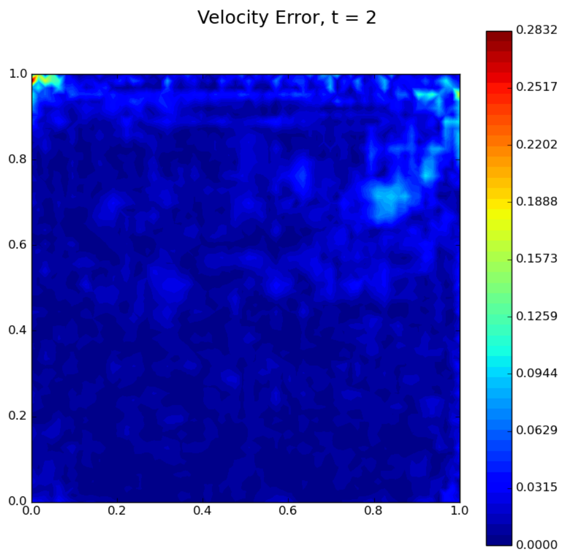

Figure C.231: Autoencoder
velocity error field for $N_H = 64 \times 64$ and Re = 500 for t = 2

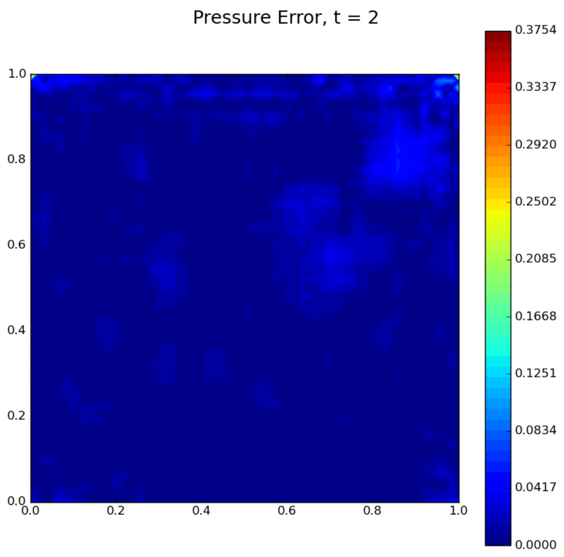

Figure C.232: Autoencoder
pressure error field for $N_H = 64 \times 64$ and Re = 500 for t = 2

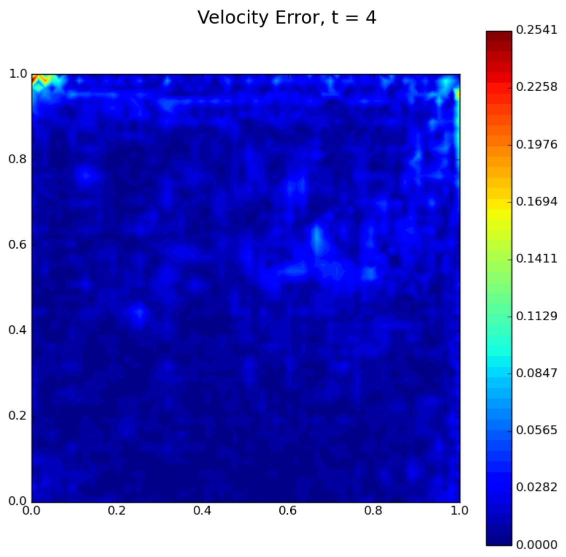

Figure C.233: Autoencoder
velocity error field for $N_H = 64 \times 64$ and Re = 500 for t = 4

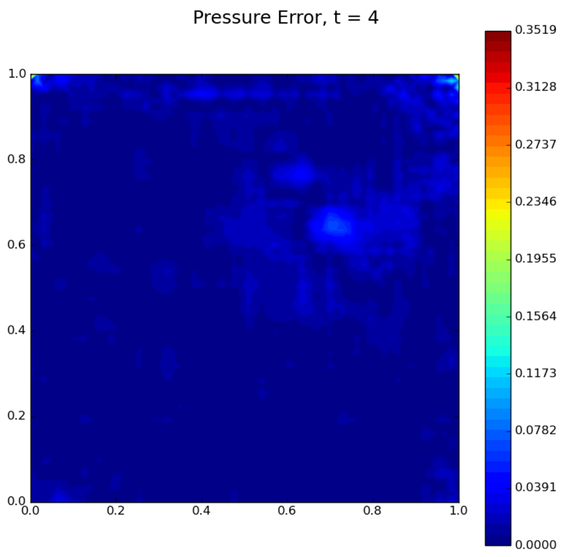

Figure C.234: Autoencoder
pressure error field for $N_H = 64 \times 64$ and Re = 500 for t = 4

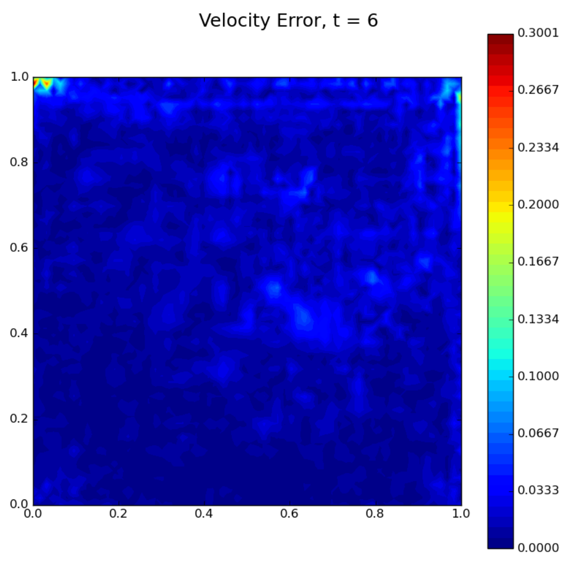

Figure C.235: Autoencoder velocity error field for $N_H = 64 \times 64$ and Re = 500 for t = 6

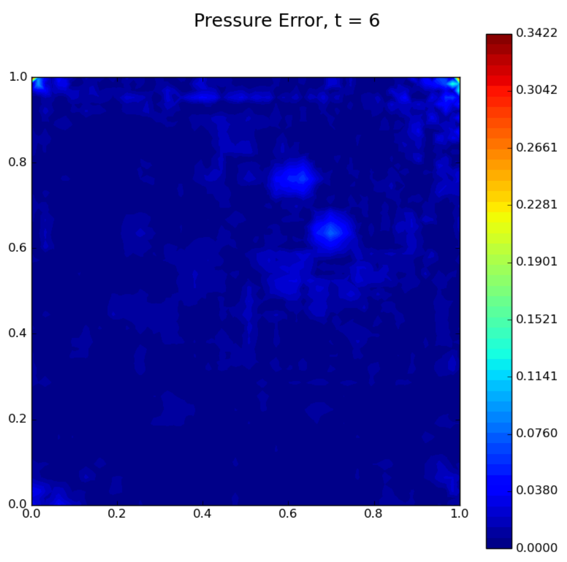

Figure C.236: Autoencoder pressure error field for $N_H = 64 \times 64$ and Re = 500 for t = 6

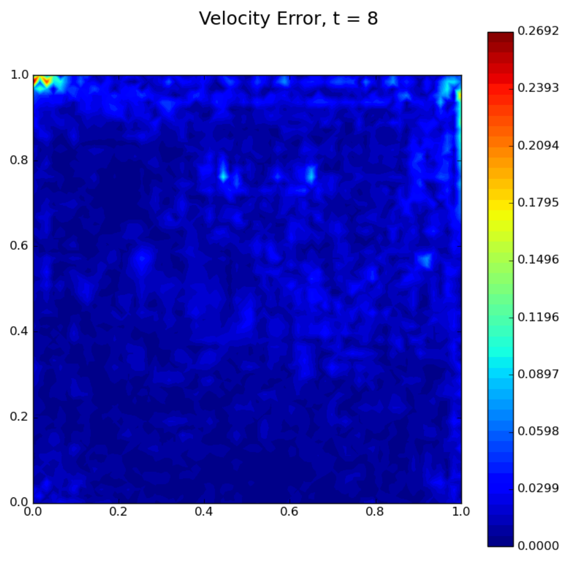

Figure C.237: Autoencoder velocity error field for $N_H = 64 \times 64$ and Re = 500 for t = 8

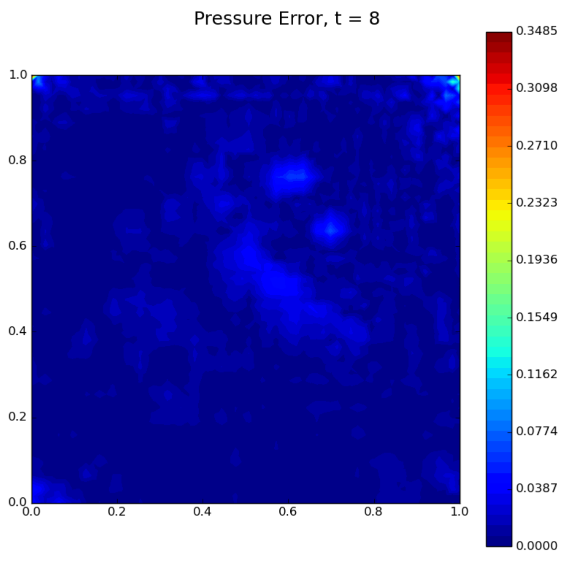

Figure C.238: Autoencoder pressure error field for $N_H = 64 \times 64$ and Re = 500 for t = 8

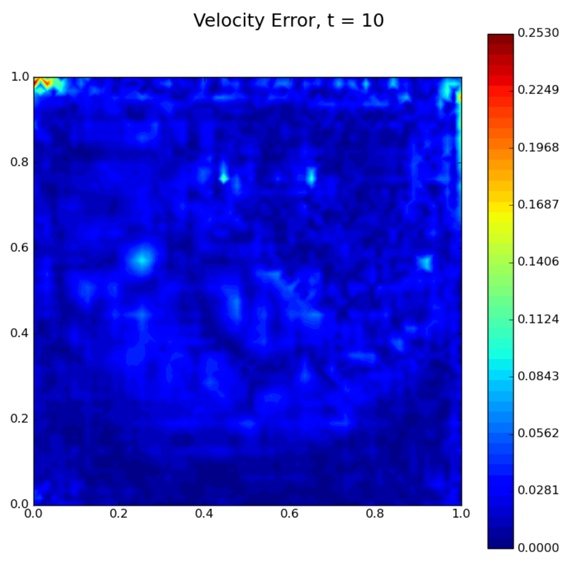

Figure C.239: Autoencoder velocity error field for $N_H = 64 \times 64$ and Re = 500 for t = 10

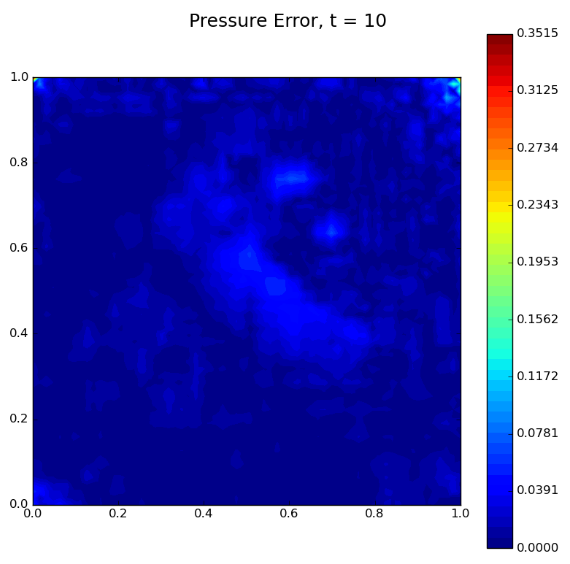

Figure C.240: Autoencoder pressure error field for $N_H = 64 \times 64$ and Re = 500 for t = 10

C.3. Lid-driven Cavity Flow Velocity Adjoint

C.3.1. Solution

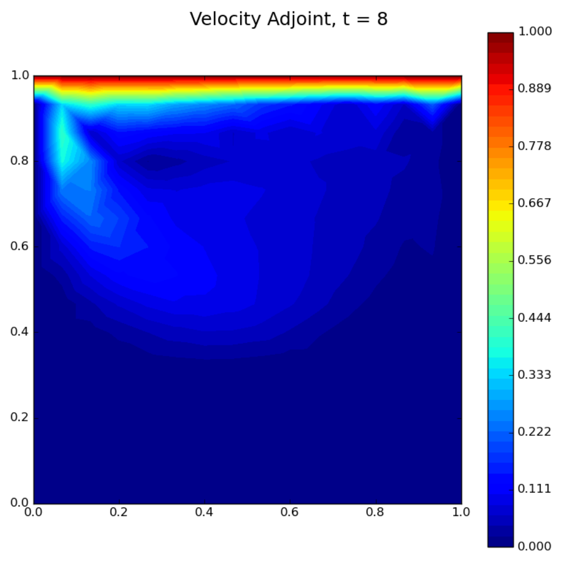

Figure C.241: Velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 8

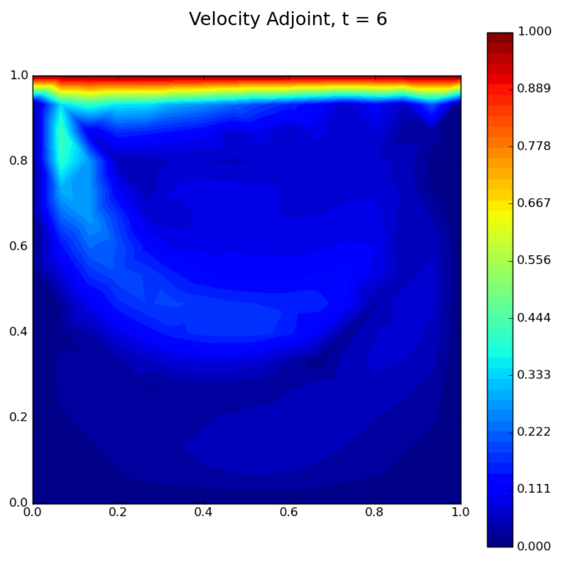

Figure C.242: Velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 6

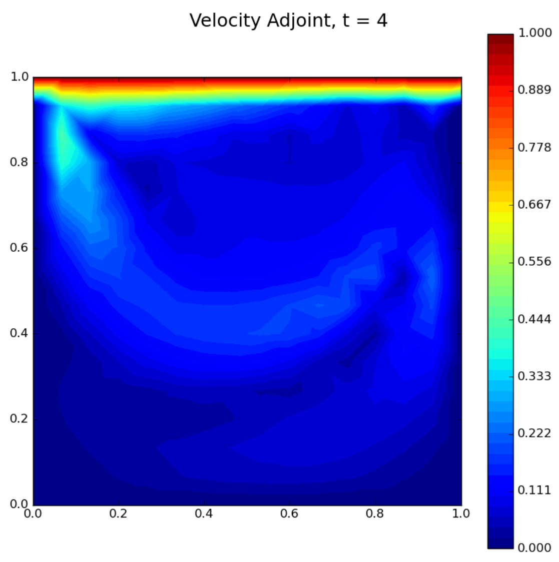

Figure C.243:
Velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 4

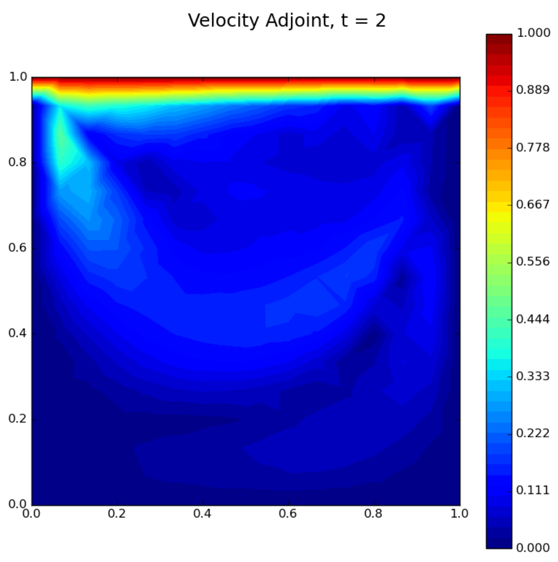

Figure C.244:
Velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 2

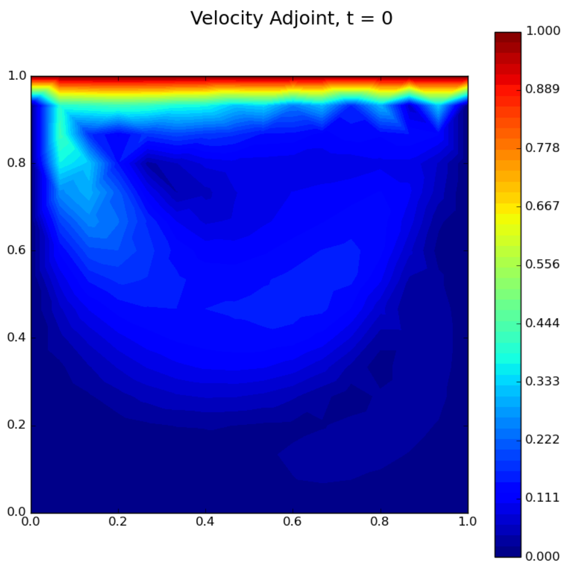

Figure C.245: Velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 0

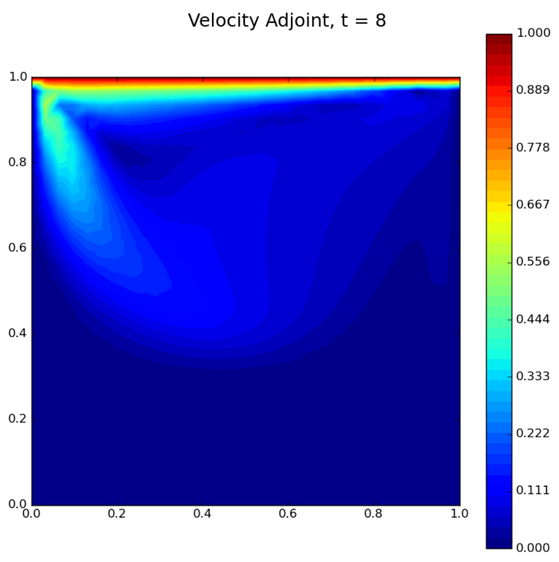

Figure C.246:
Velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 8

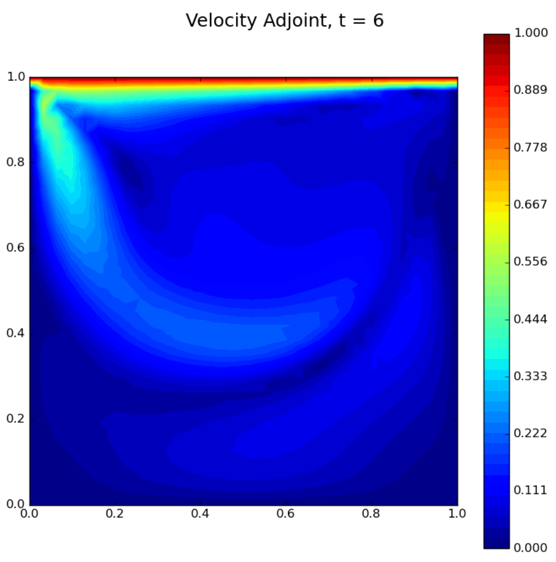

Figure C.247:
Velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 6

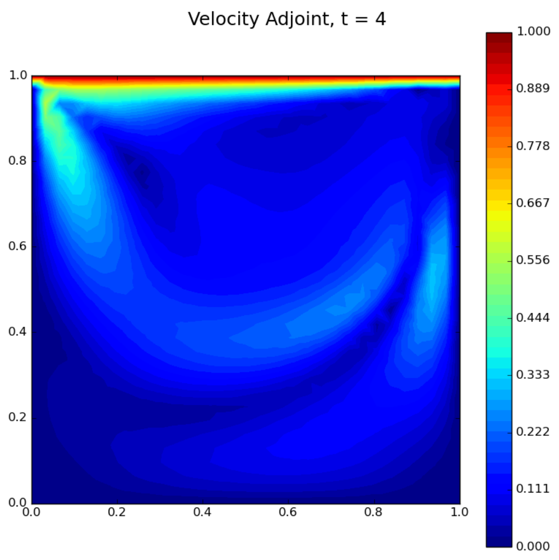

Figure C.248:
Velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 4

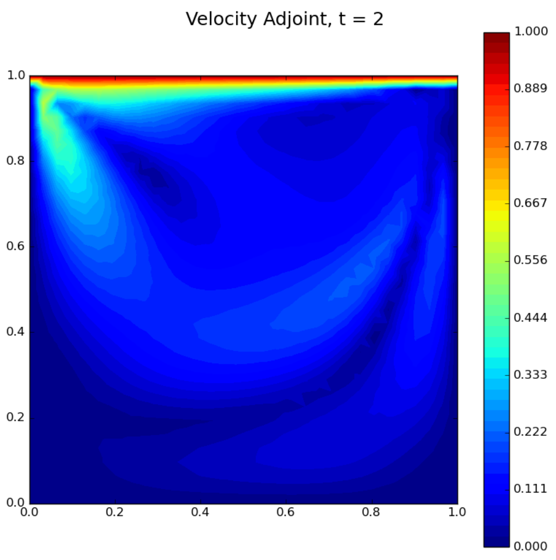

Figure C.249:
Velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 2

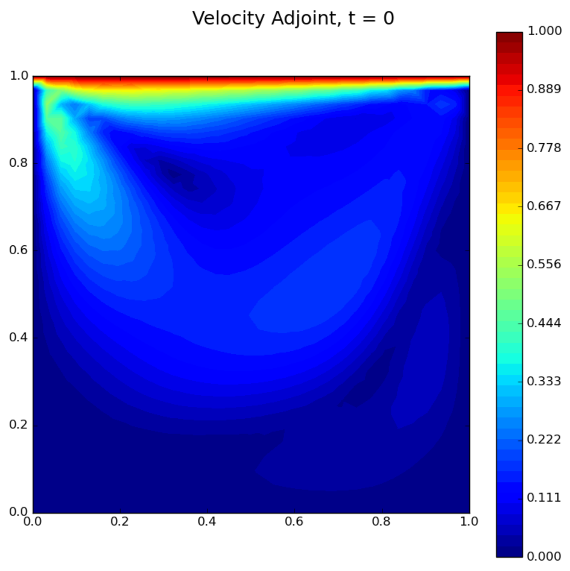

Figure C.250: Velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 0

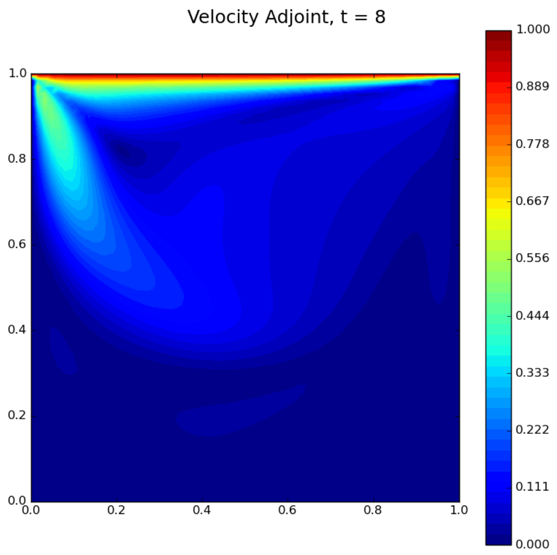

Figure C.251:
Velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 8

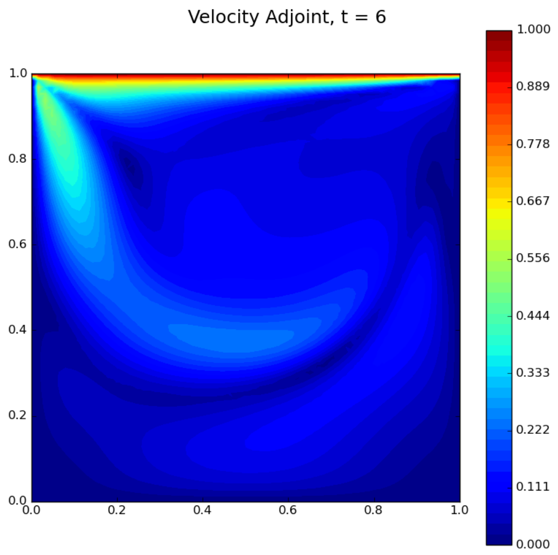

Figure C.252:
Velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 6

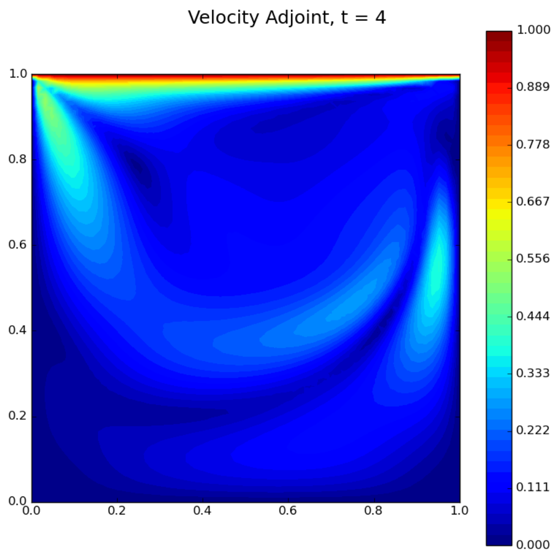

Figure C.253:
Velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 4

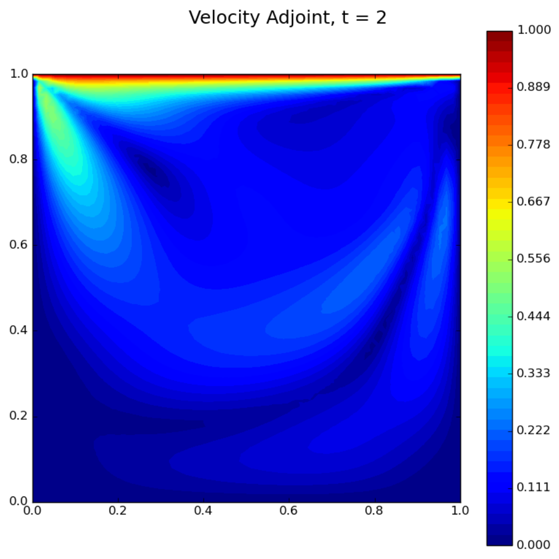

Figure C.254:
Velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 2

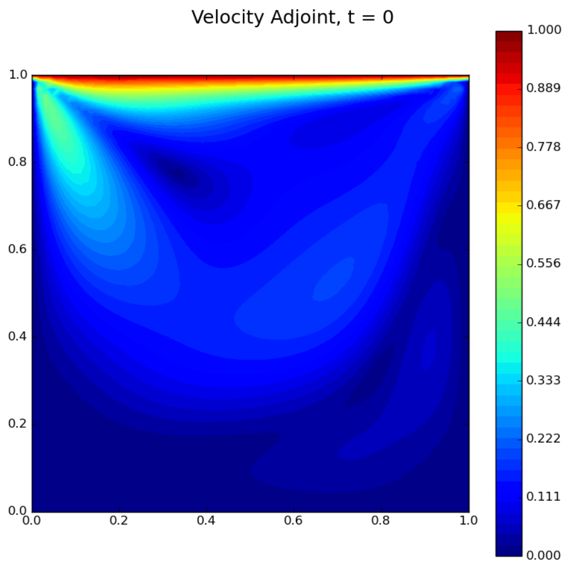

Figure C.255: Velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 0

C.3.2. POD

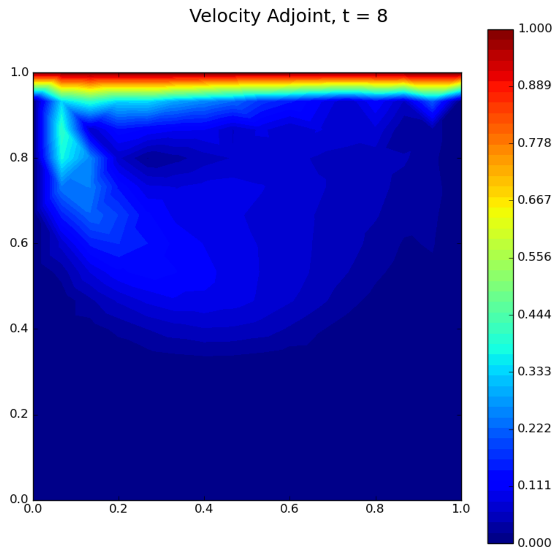

Figure C.256:
POD velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 8

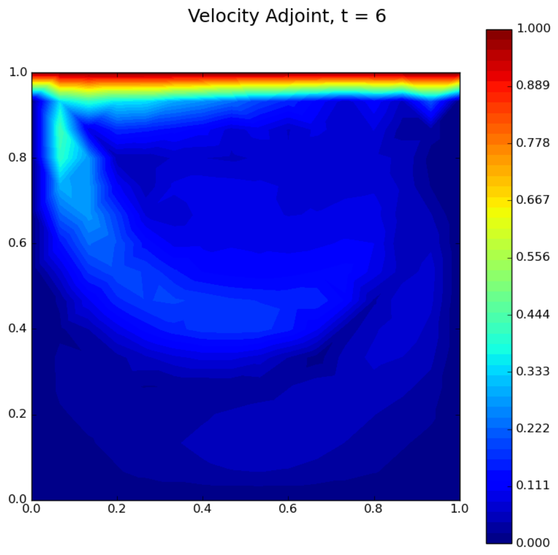

Figure C.257:
POD velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 6

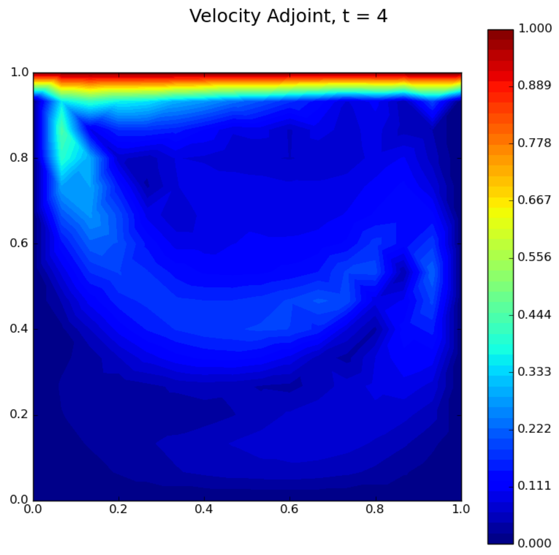

Figure C.258:
POD velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 4

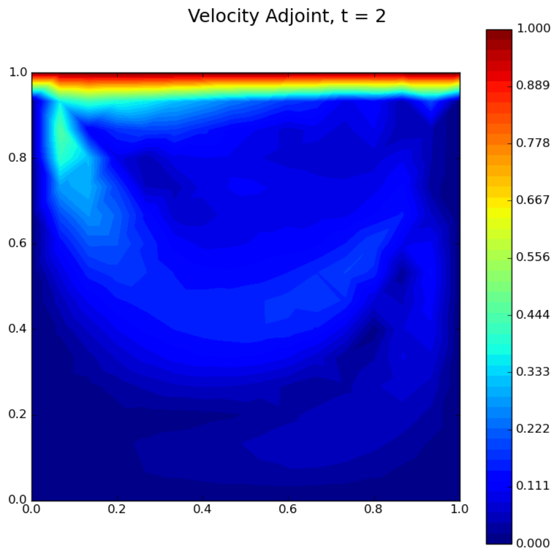

Figure C.259:
POD velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 2

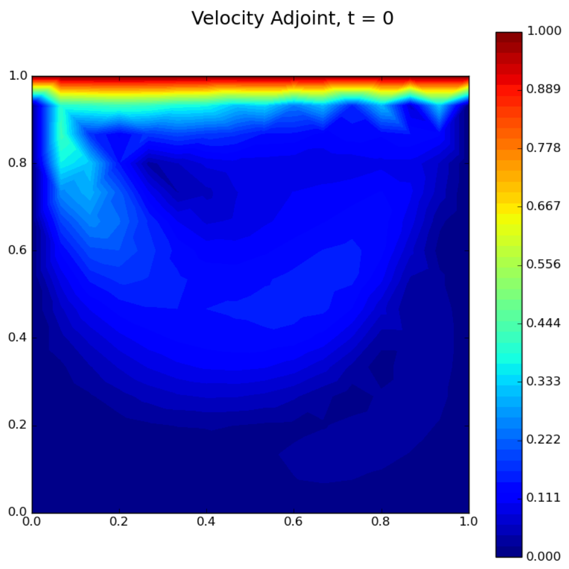

Figure C.260: POD velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 0

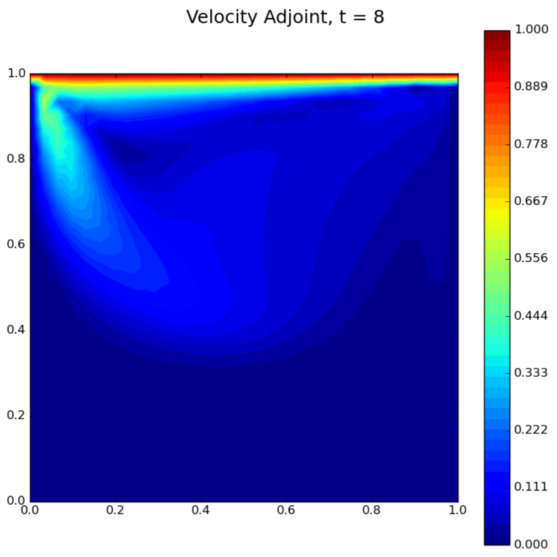

Figure C.261:
POD velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 8

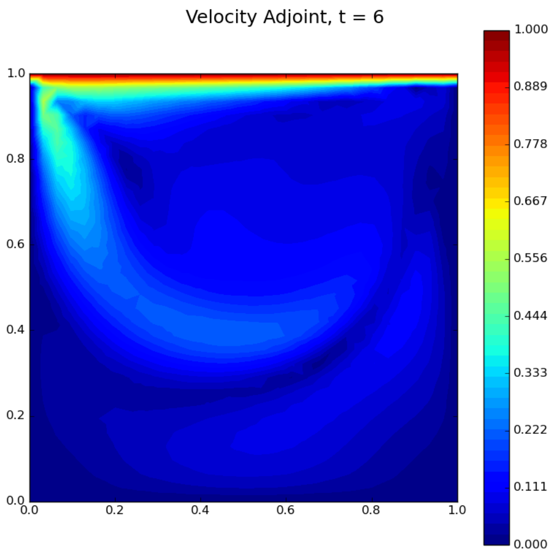

Figure C.262:
POD velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 6

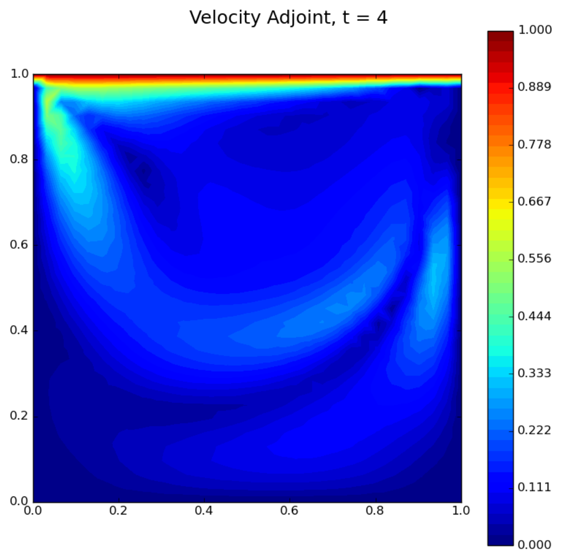

Figure C.263:
POD velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 4

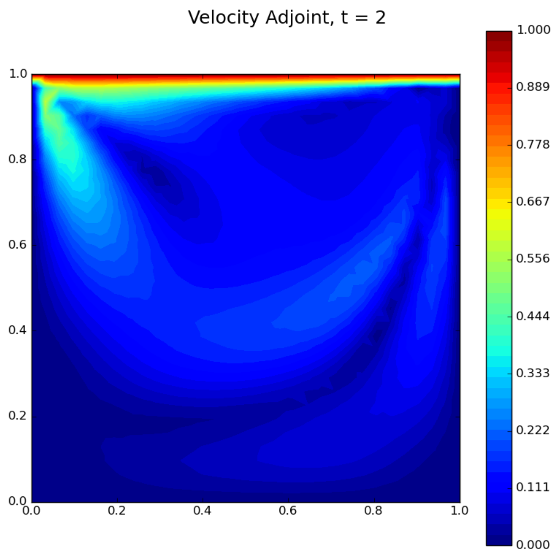

Figure C.264:
POD velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 2

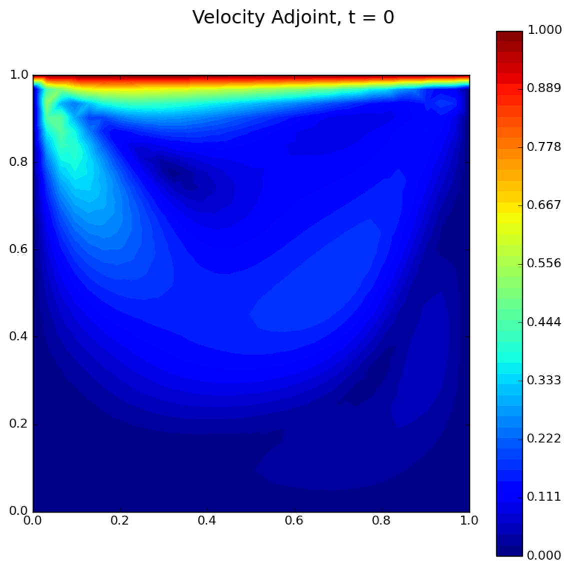

Figure C.265: POD velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 0

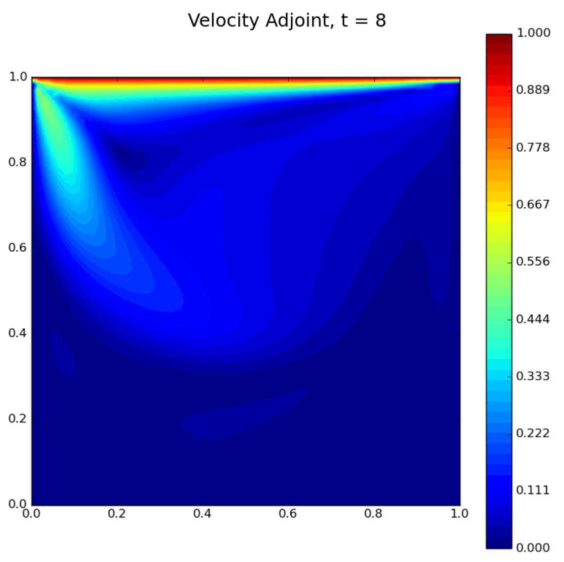

Figure C.266:
POD velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 8

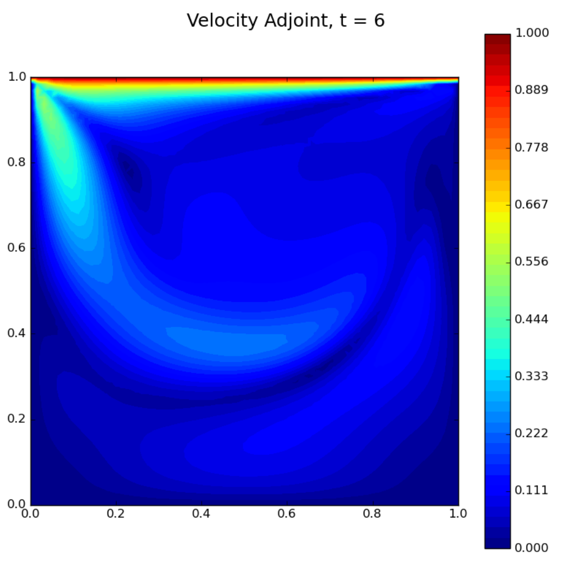

Figure C.267:
POD velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 6

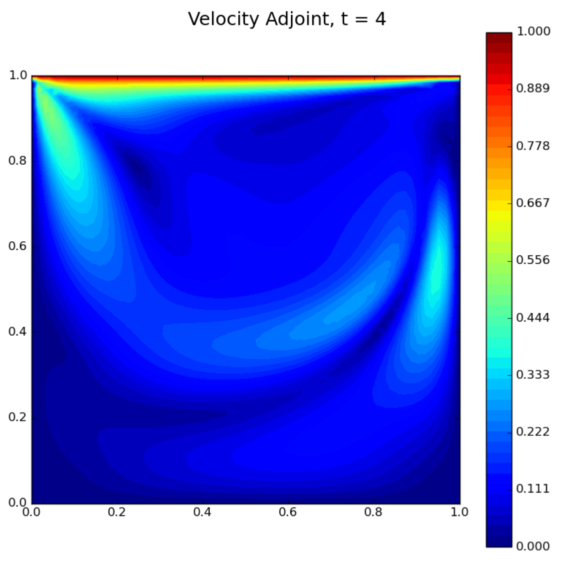

Figure C.268:
POD velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 4

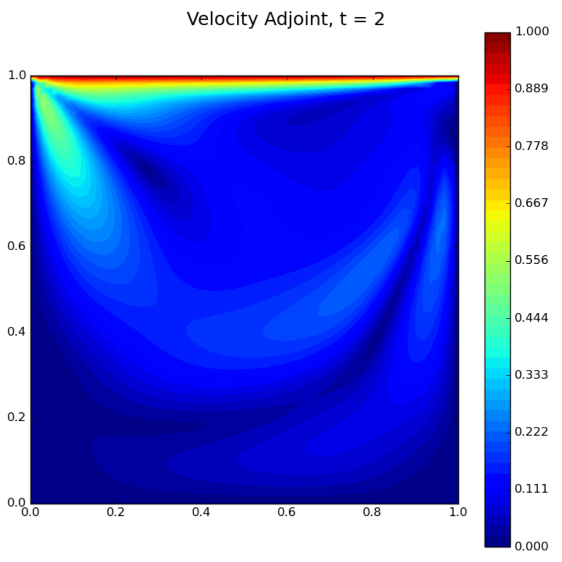

Figure C.269:
POD velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 2

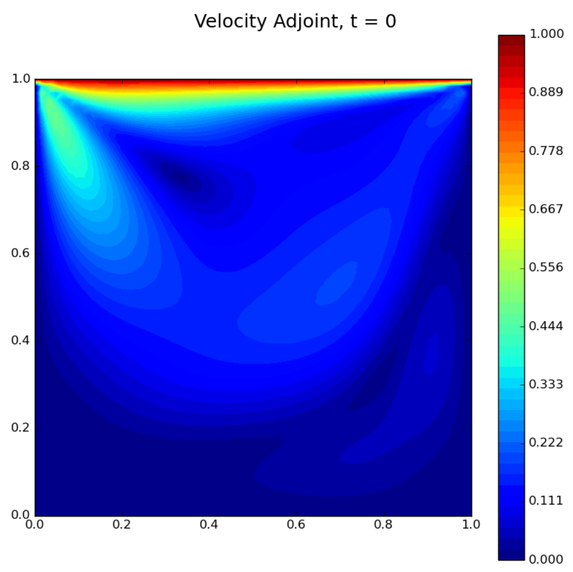

Figure C.270: POD velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 0

C.3.3. Autoencoder

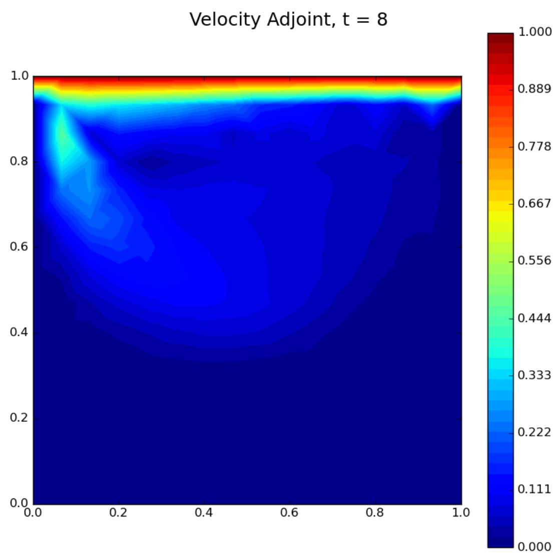

Figure C.271: Autoencoder velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 8

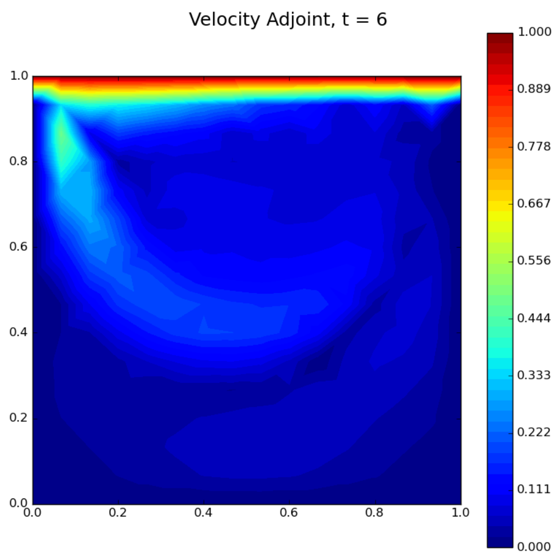

Figure C.272: Autoencoder velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 6

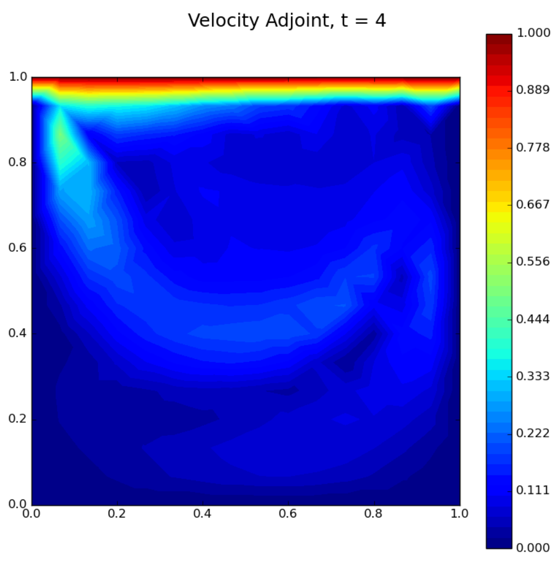

Figure C.273: Autoencoder velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 4

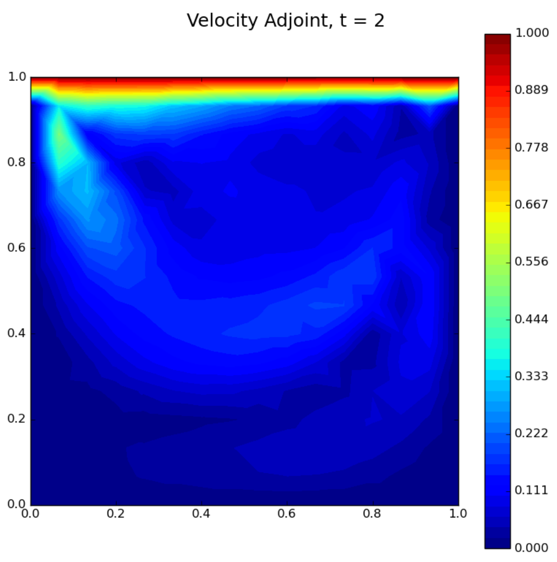

Figure C.274: Autoencoder velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 2

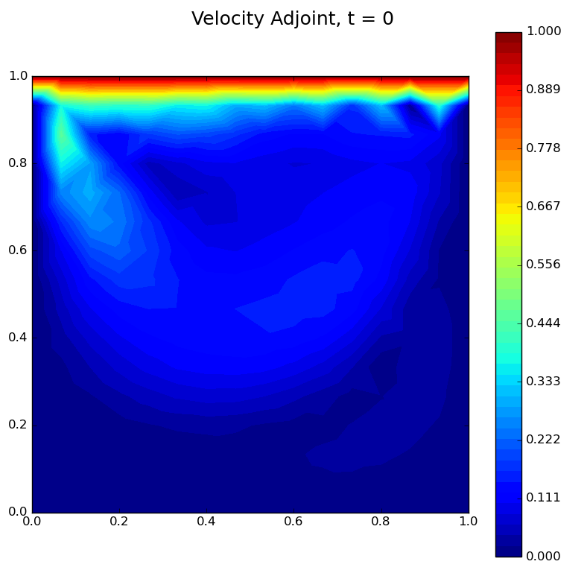

Figure C.275: Autoencoder velocity adjoint for $N_h = 16 \times 16$ and Re = 500 for t = 0

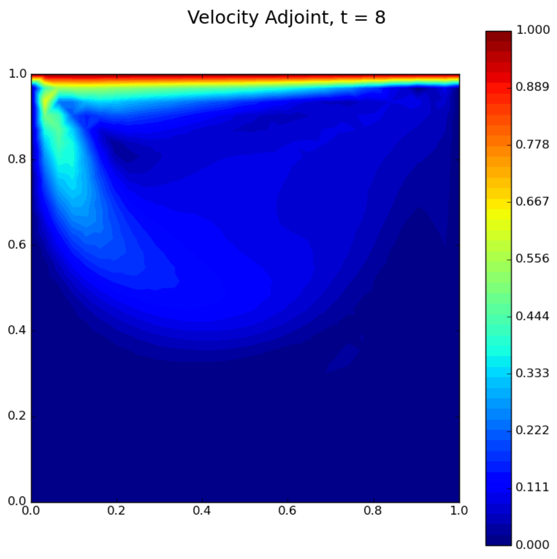

Figure C.276: Autoencoder velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 8

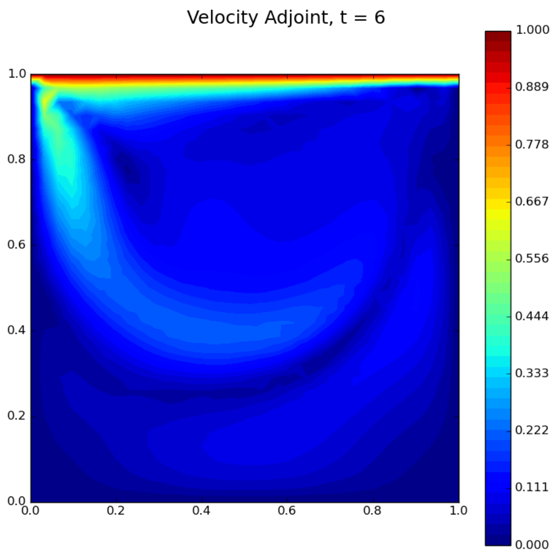

Figure C.277: Autoencoder velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 6

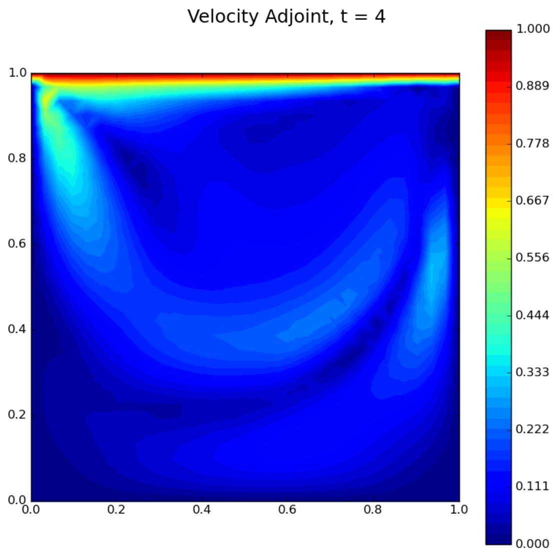

Figure C.278: Autoencoder velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 4

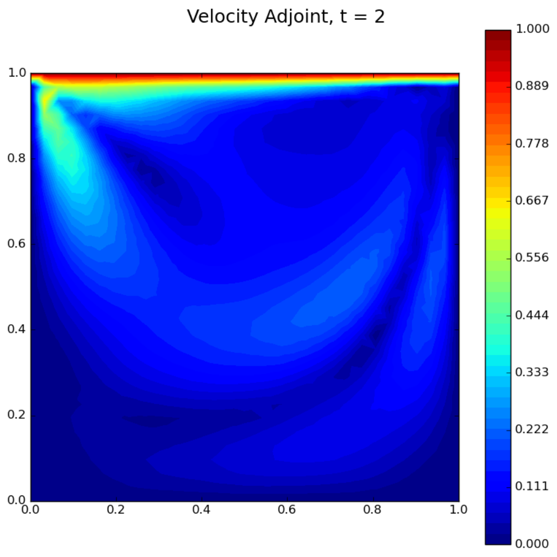

Figure C.279: Autoencoder velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 2

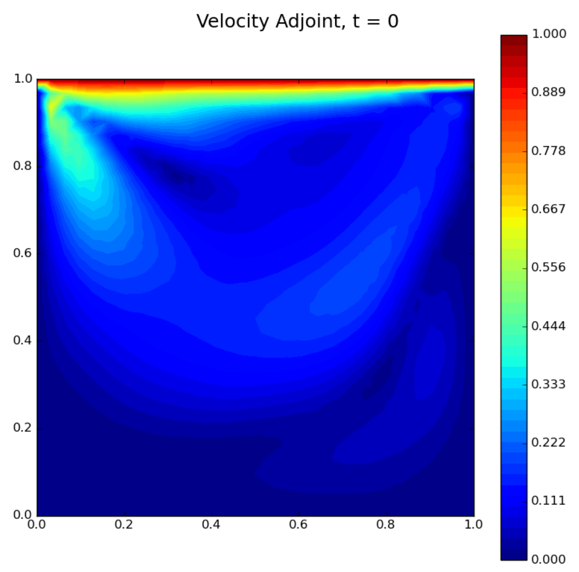

Figure C.280: Autoencoder velocity adjoint for $N_h = 32 \times 32$ and Re = 500 for t = 0

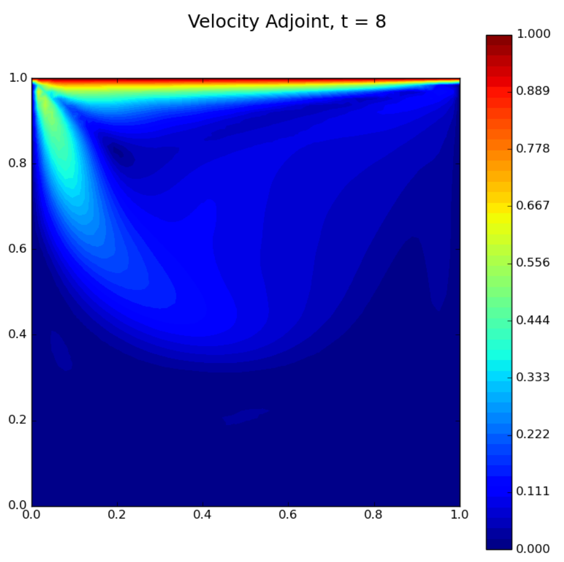

Figure C.281: Autoencoder velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 8

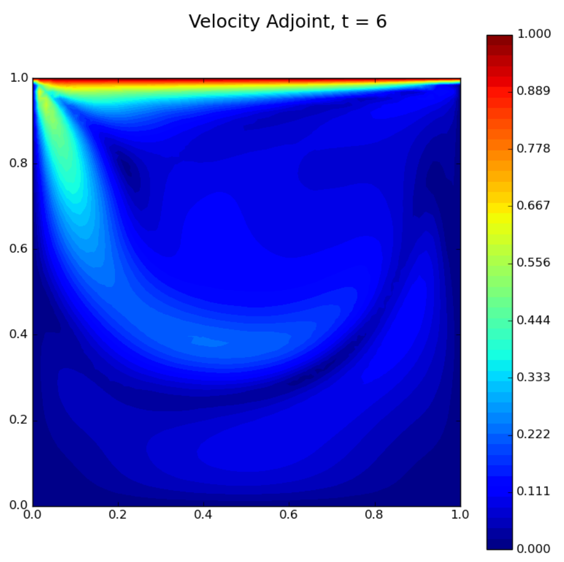

Figure C.282: Autoencoder velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 6

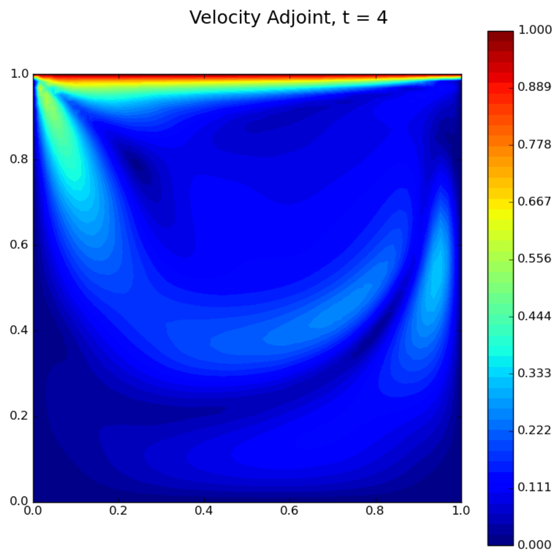

Figure C.283: Autoencoder velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 4

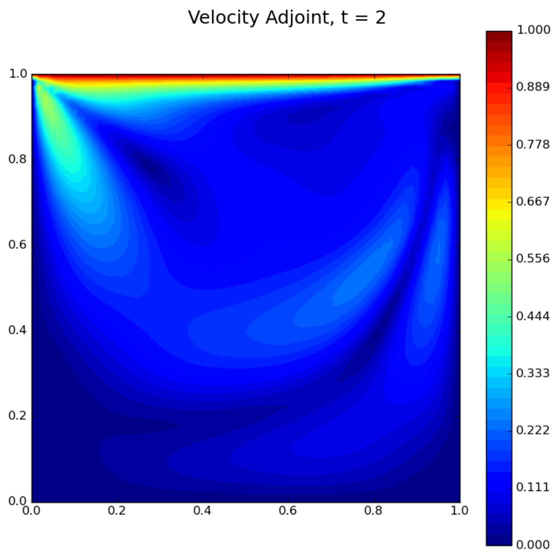

Figure C.284: Autoencoder velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 2

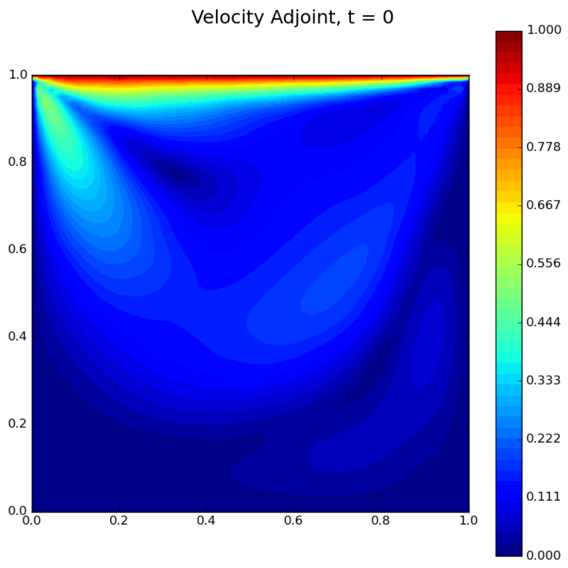

Figure C.285: Autoencoder velocity adjoint for $N_h = 64 \times 64$ and Re = 500 for t = 0

C.4. Lid-driven Cavity Flow Velocity Adjoint Error

C.4.1. POD

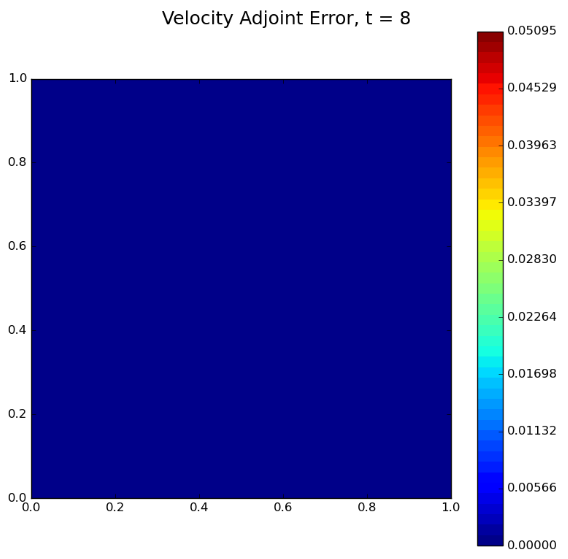

Figure C.286: POD
velocity adjoint error for $N_h = 16 \times 16$ and Re = 500 for t = 8

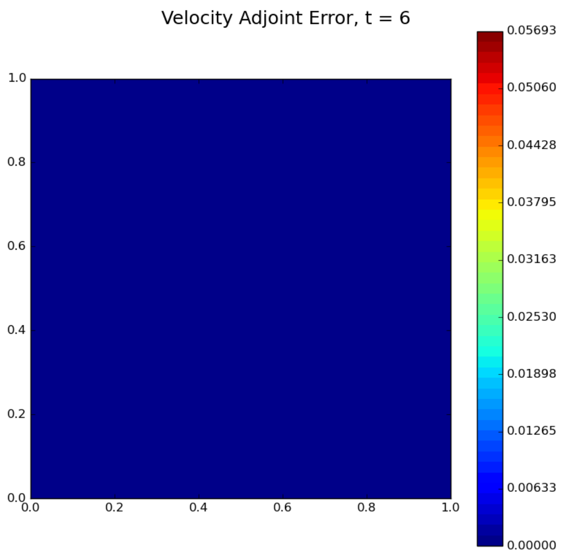

Figure C.287: POD
velocity adjoint error for $N_h = 16 \times 16$ and Re = 500 for t = 6

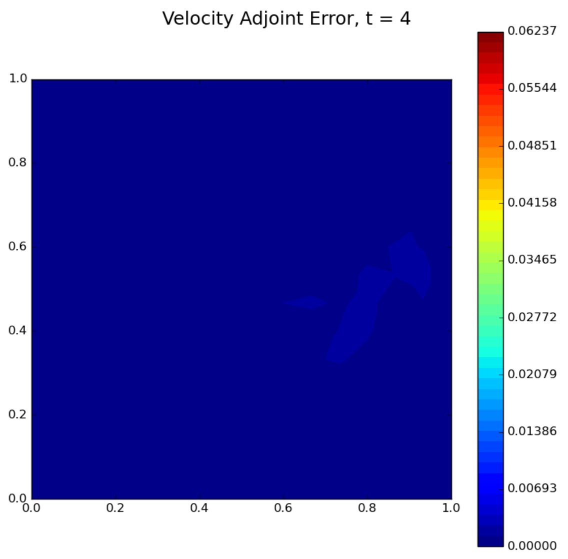

Figure C.288: POD
velocity adjoint error for $N_h = 16 \times 16$ and Re = 500 for t = 4

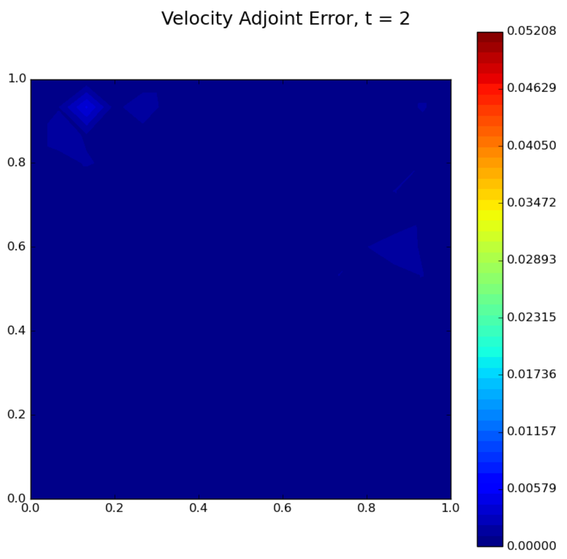

Figure C.289: POD
velocity adjoint error for $N_h = 16 \times 16$ and Re = 500 for t = 2

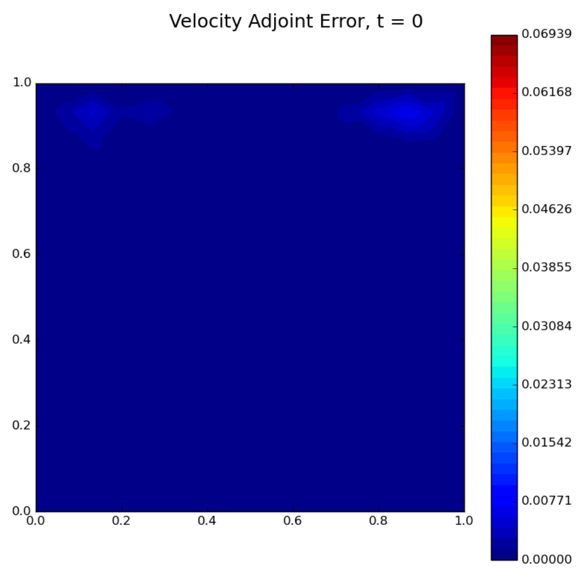

Figure C.290: POD velocity adjoint error for $N_h = 16 \times 16$ and Re = 500 for t = 0

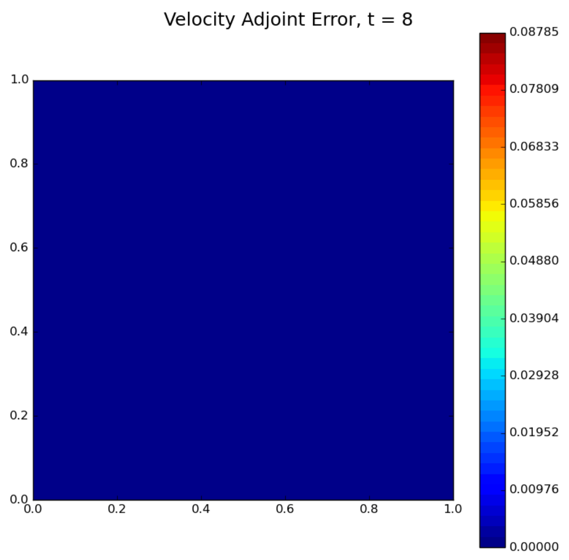

Figure C.291: POD velocity adjoint error for $N_h = 32 \times 32$ and Re = 500 for t = 8

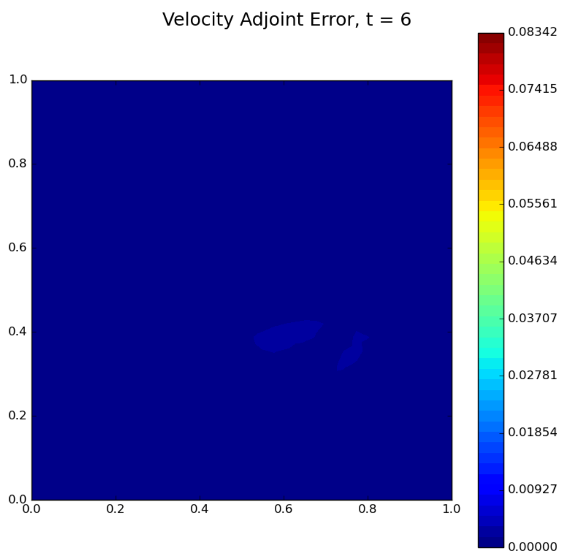

Figure C.292: POD velocity adjoint error for $N_h = 32 \times 32$ and Re = 500 for t = 6

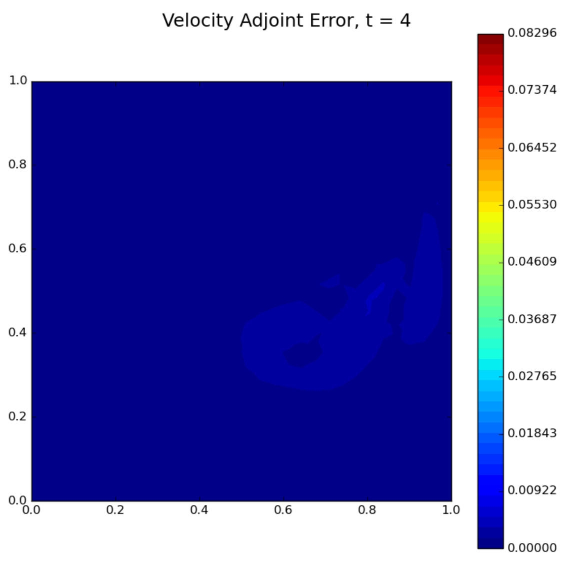

Figure C.293: POD velocity adjoint error for $N_h = 32 \times 32$ and Re = 500 for t = 4

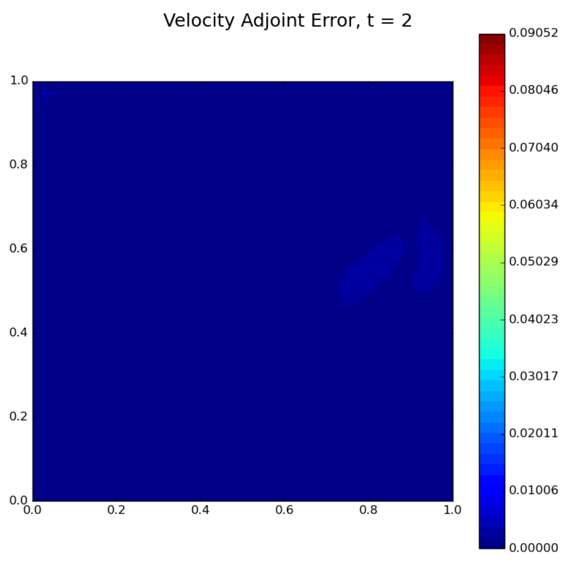

Figure C.294: POD velocity adjoint error for $N_h = 32 \times 32$ and Re = 500 for t = 2

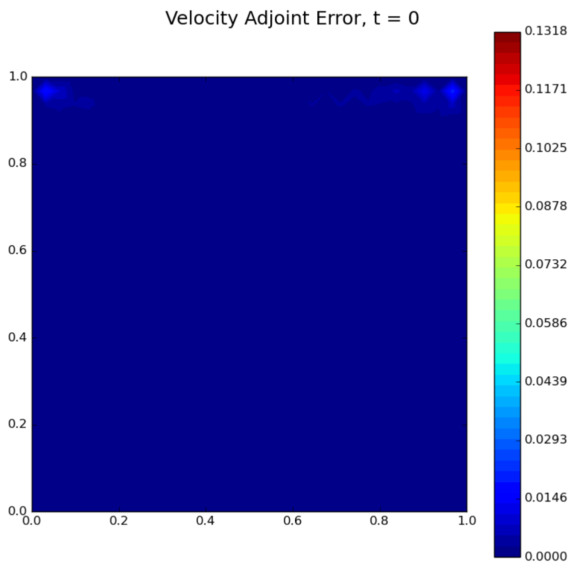

Figure C.295: POD velocity adjoint error for $N_h = 32 \times 32$ and Re = 500 for t = 0

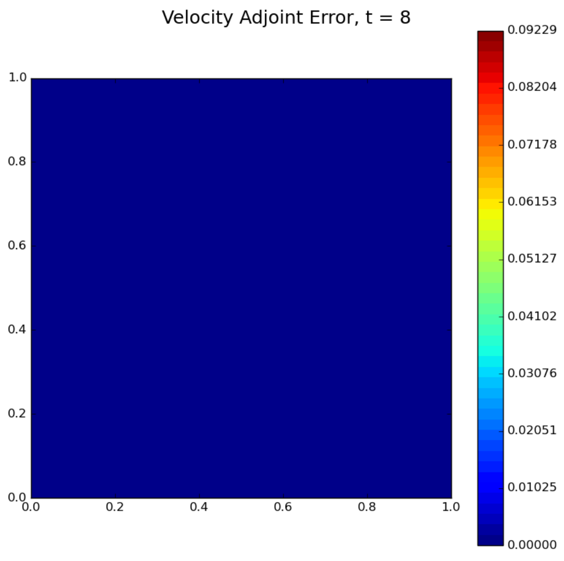

Figure C.296: POD velocity adjoint error for $N_h = 64 \times 64$ and Re = 500 for t = 8

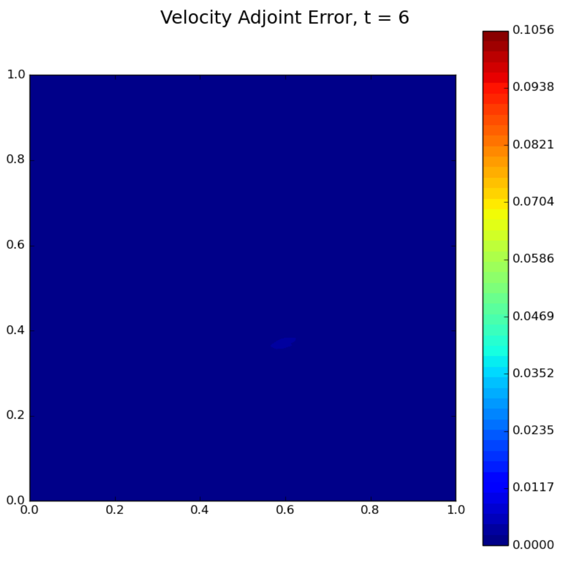

Figure C.297: POD velocity adjoint error for $N_h = 64 \times 64$ and Re = 500 for t = 6

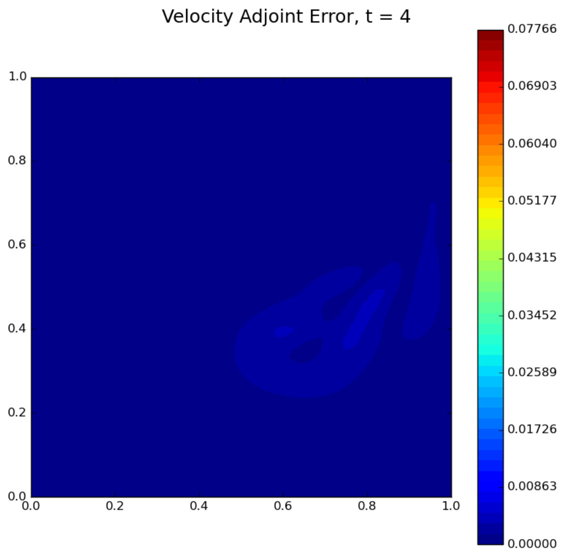

Figure C.298: POD velocity adjoint error for $N_h = 64 \times 64$ and Re = 500 for t = 4

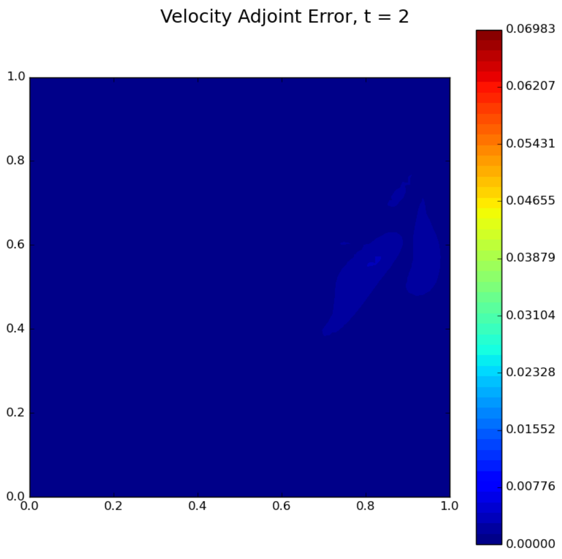

Figure C.299: POD velocity adjoint error for $N_h = 64 \times 64$ and Re = 500 for t = 2

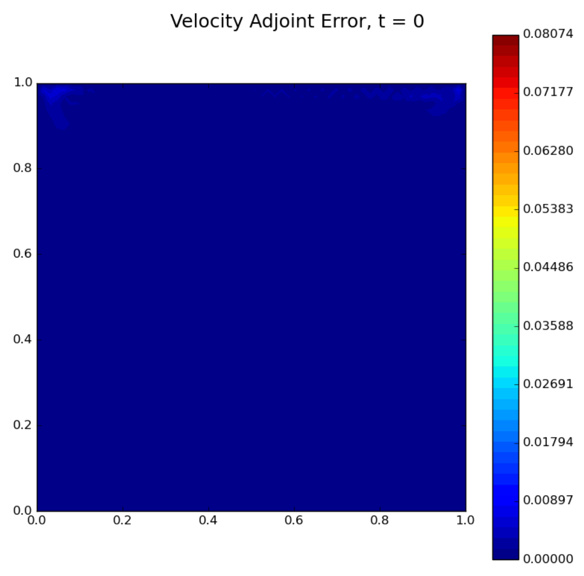

Figure C.300: POD velocity adjoint error for $N_h = 64 \times 64$ and Re = 500 for t = 0

C.4.2. Autoencoder

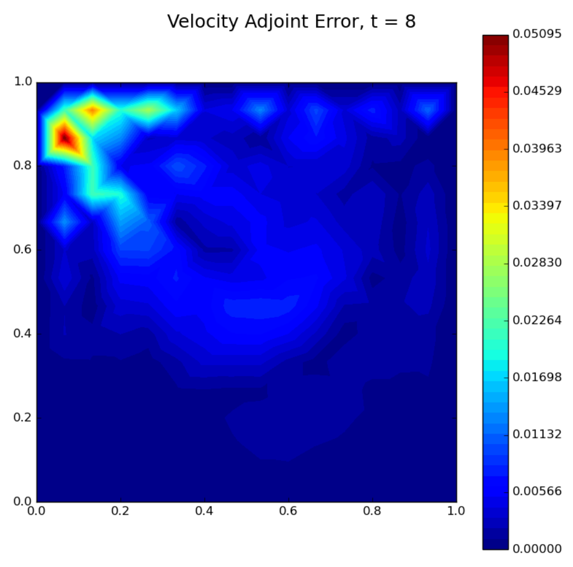

Figure C.301: Autoencoder velocity adjoint error for $N_h = 16 \times 16$ and Re = 500 for t = 8

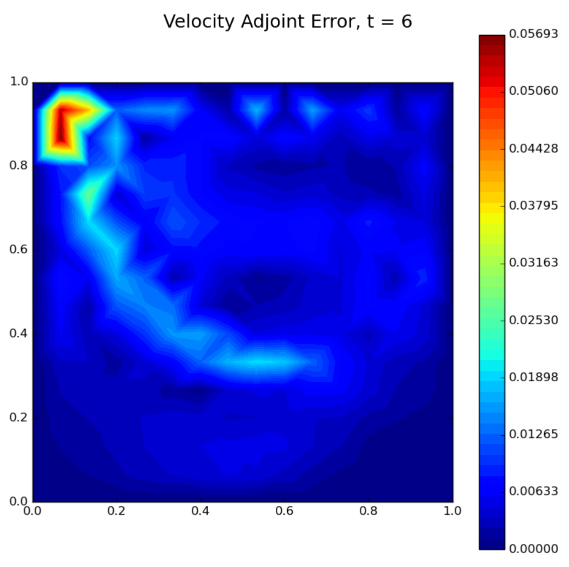

Figure C.302: Autoencoder velocity adjoint error for $N_h = 16 \times 16$ and Re = 500 for t = 6

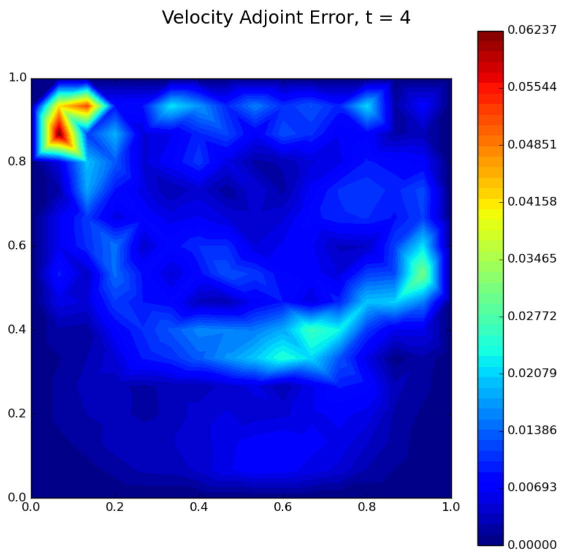

Figure C.303: Autoencoder velocity adjoint error for $N_h = 16 \times 16$ and Re = 500 for t = 4

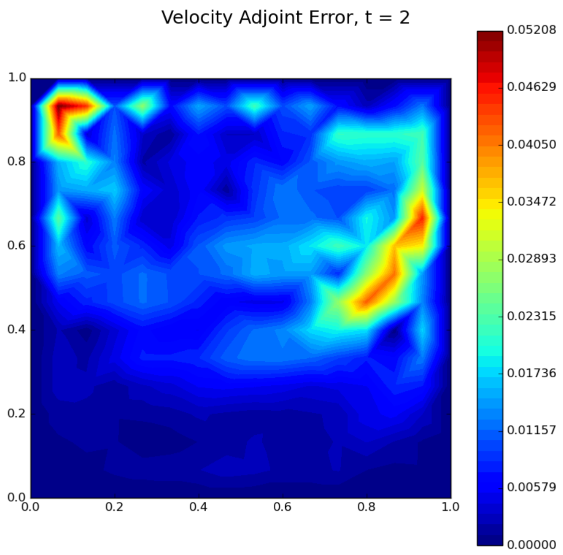

Figure C.304: Autoencoder velocity adjoint error for $N_h = 16 \times 16$ and Re = 500 for t = 2

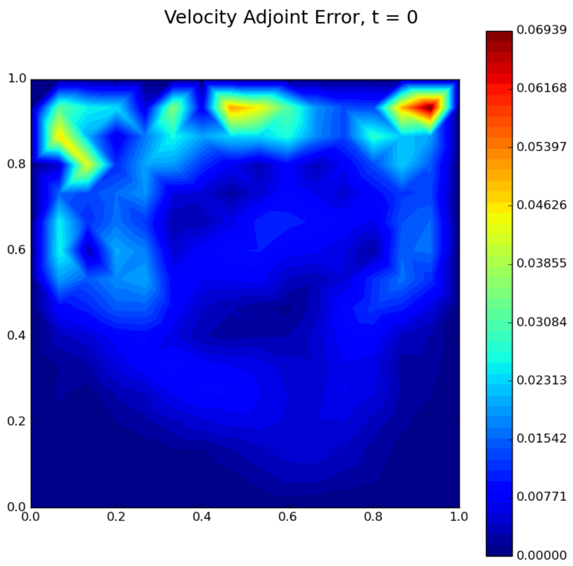

Figure C.305: Autoencoder velocity adjoint error for $N_h = 16 \times 16$ and Re = 500 for t = 0

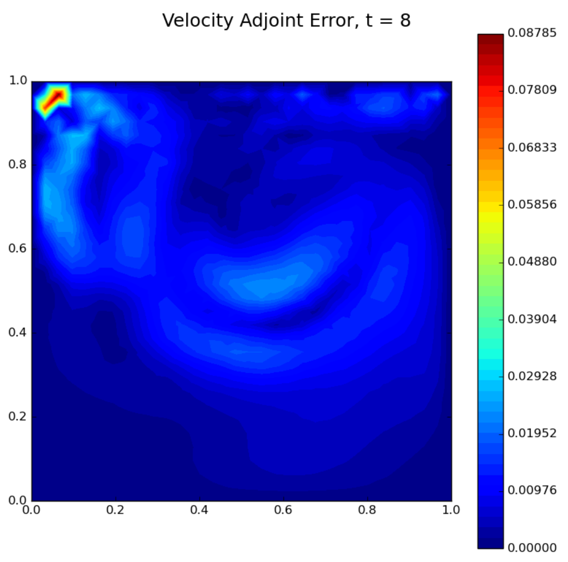

Figure C.306: Autoencoder velocity adjoint error for $N_h = 32 \times 32$ and Re = 500 for t = 8

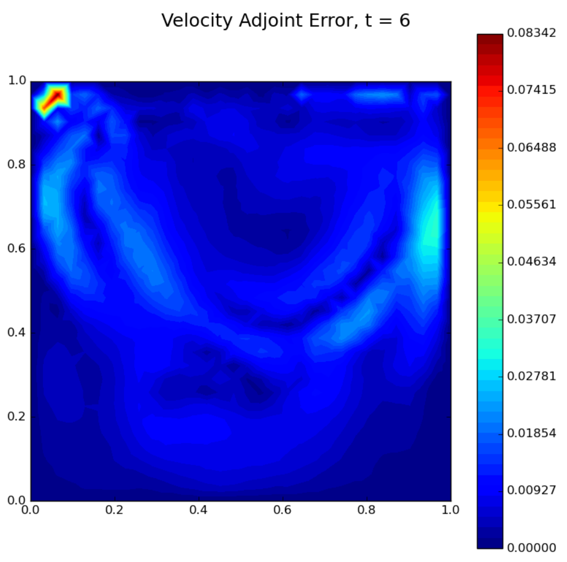

Figure C.307: Autoencoder velocity adjoint error for $N_h = 32 \times 32$ and Re = 500 for t = 6

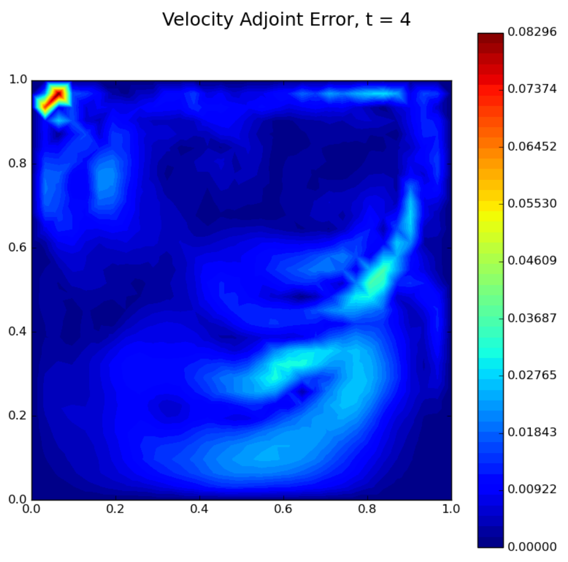

Figure C.308: Autoencoder velocity adjoint error for $N_h = 32 \times 32$ and Re = 500 for t = 4

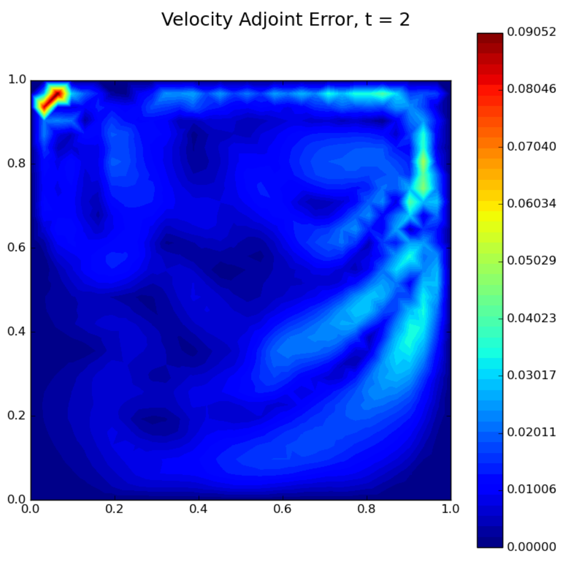

Figure C.309: Autoencoder velocity adjoint error for $N_h = 32 \times 32$ and Re = 500 for t = 2

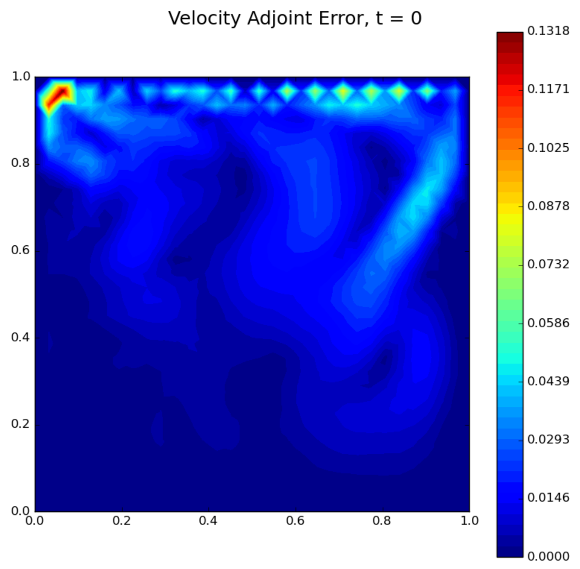

Figure C.310: Autoencoder velocity adjoint error for $N_h = 32 \times 32$ and Re = 500 for t = 0

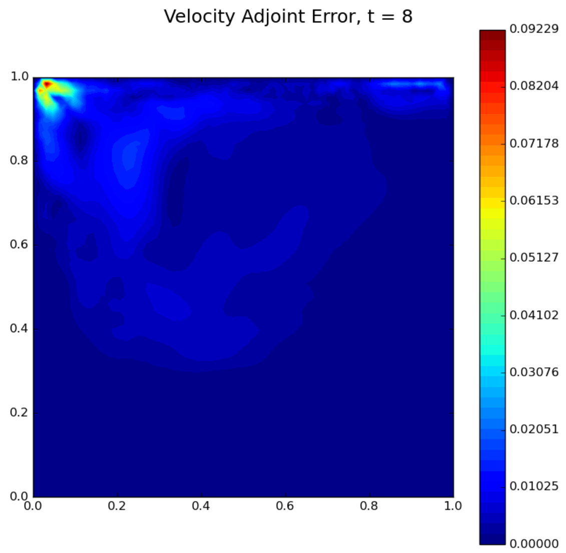

Figure C.311: Autoencoder velocity adjoint error for $N_h = 64 \times 64$ and Re = 500 for t = 8

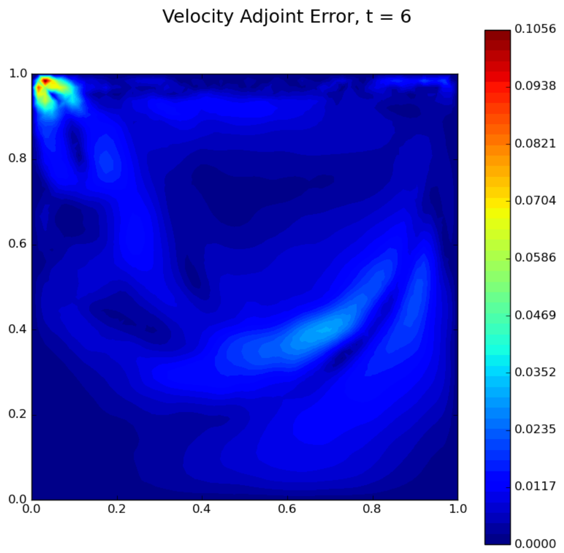

Figure C.312: Autoencoder velocity adjoint error for $N_h = 64 \times 64$ and Re = 500 for t = 6

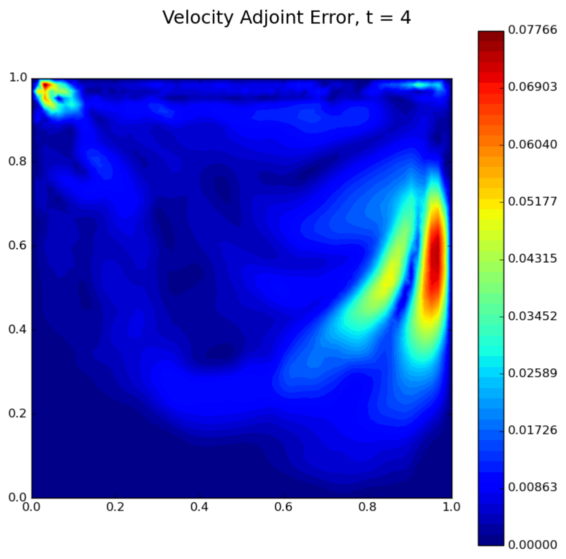

Figure C.313: Autoencoder velocity adjoint error for $N_h = 64 \times 64$ and Re = 500 for t = 4

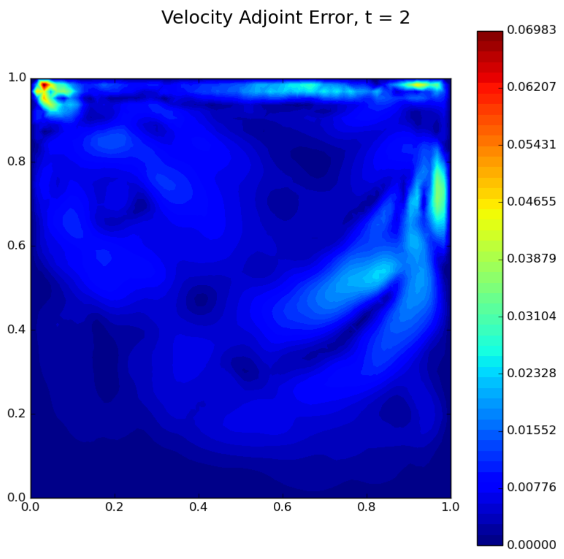

Figure C.314: Autoencoder velocity adjoint error for $N_h = 64 \times 64$ and Re = 500 for t = 2

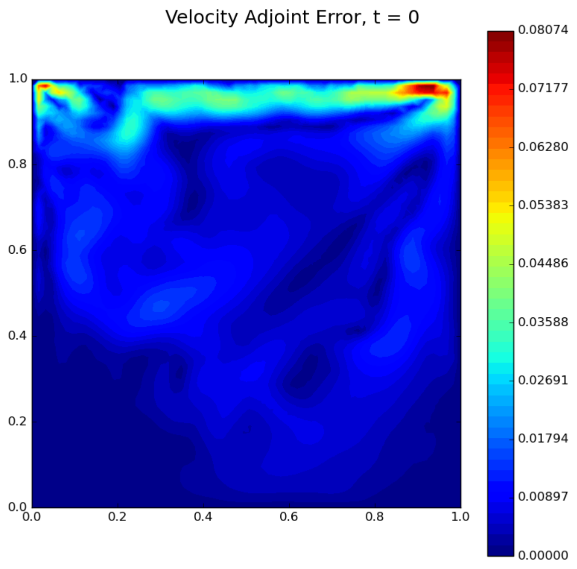

Figure C.315: Autoencoder velocity adjoint error for $N_h = 64 \times 64$ and Re = 500 for t = 0

C.5. Lid-driven Cavity Flow Adjoint Weighted Residual Solution

C.5.1. Solution

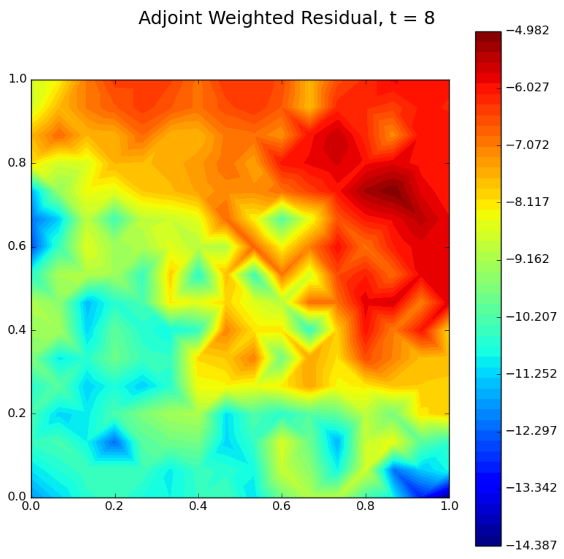

Figure C.316: Adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 8

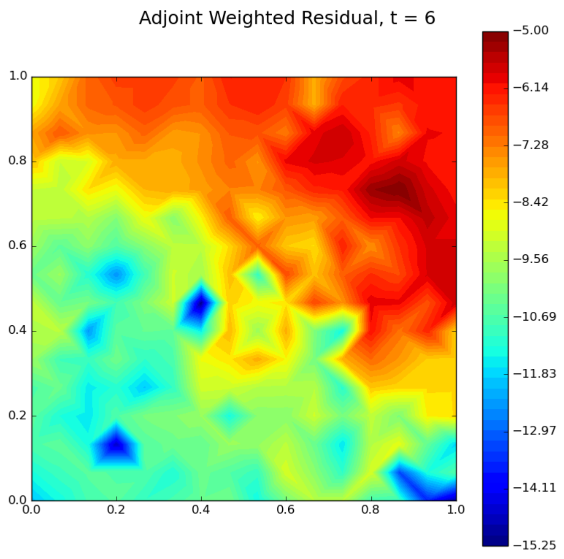

Figure C.317: Adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 6

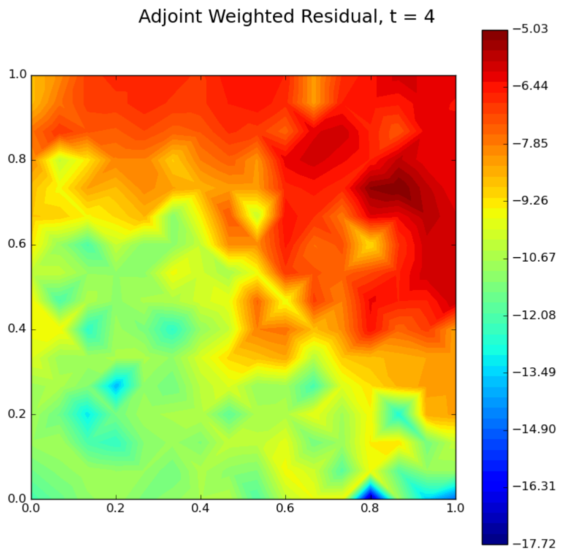

Figure C.318: Adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 4

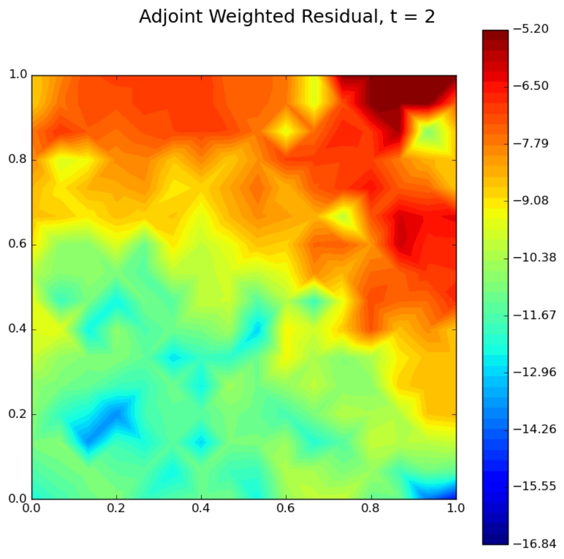

Figure C.319: Adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 2

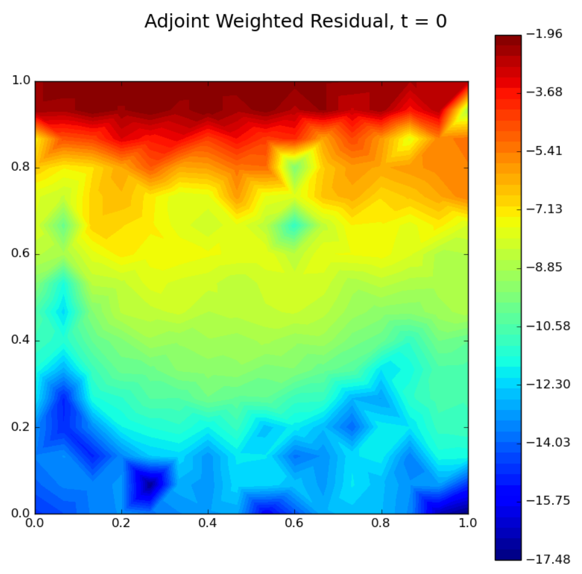

Figure C.320: Adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 0

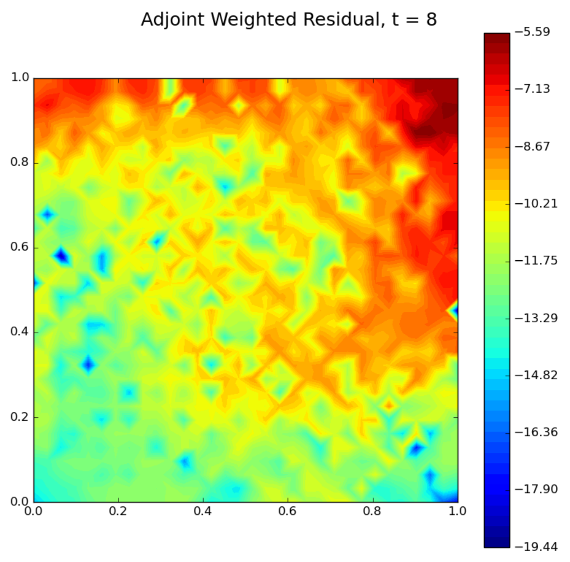

Figure C.321: Adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 8

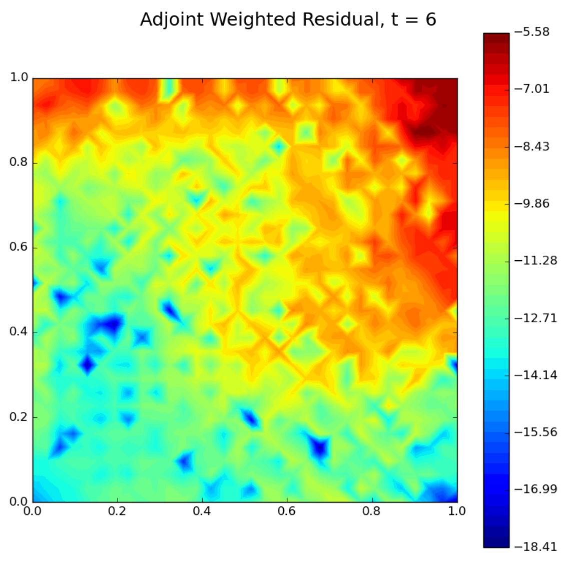

Figure C.322: Adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 6

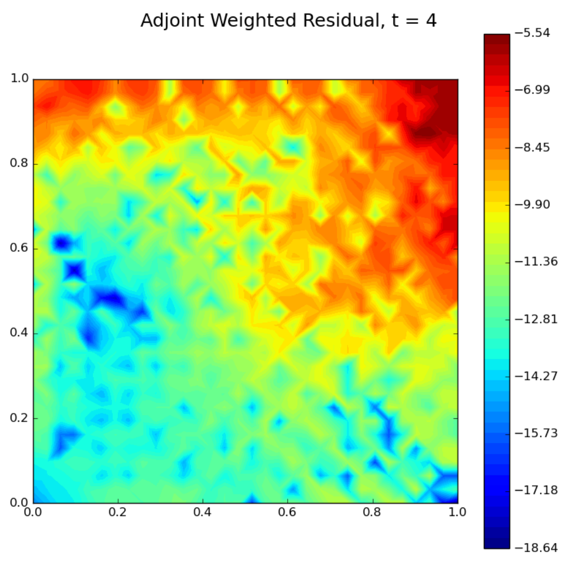

Figure C.323: Adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 4

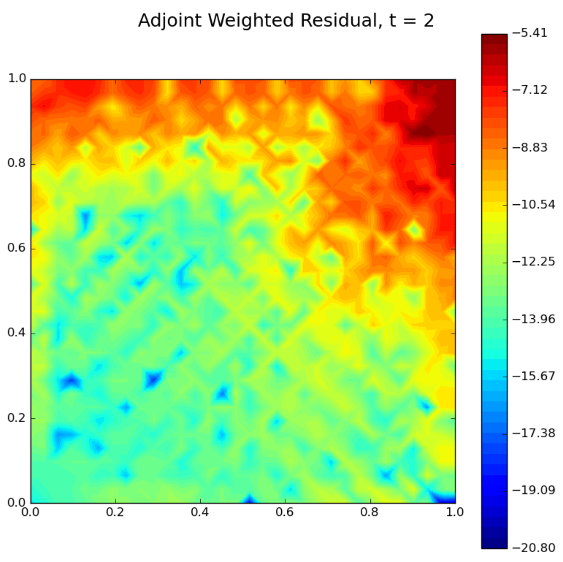

Figure C.324: Adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 2

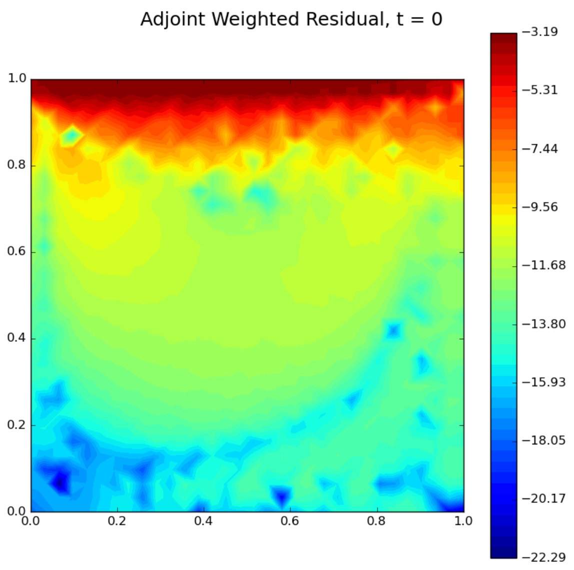

Figure C.325: Adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 0

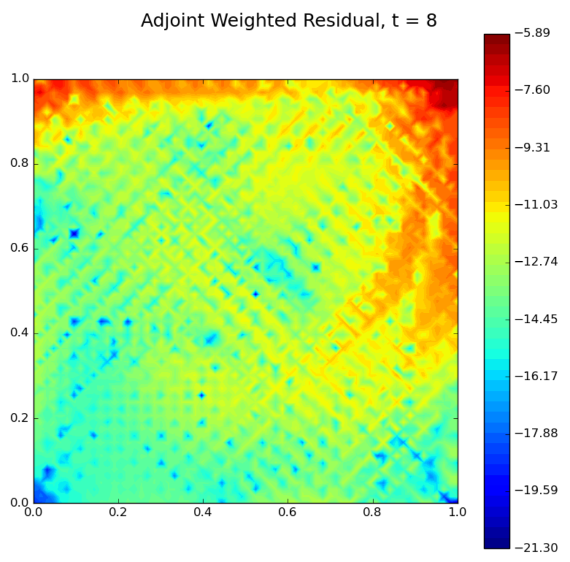

Figure C.326: Adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 8

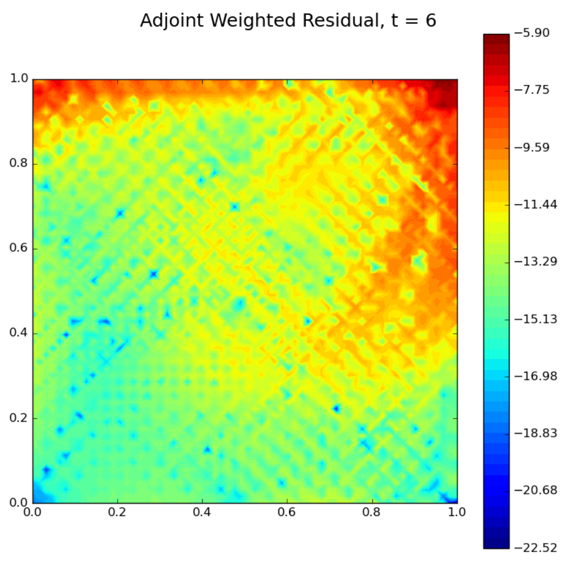

Figure C.327: Adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 6

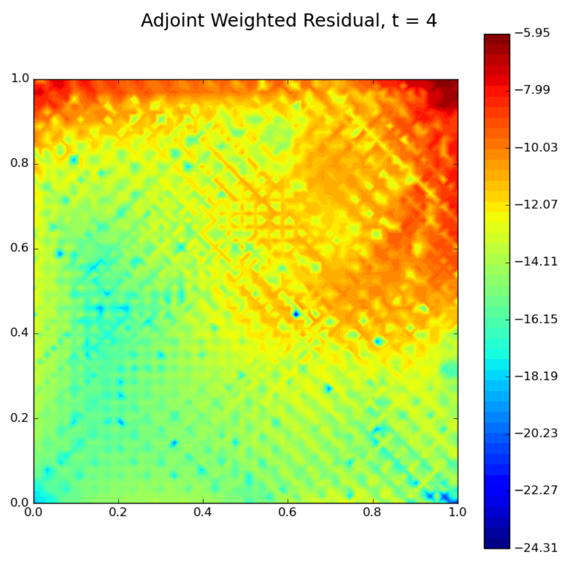

Figure C.328: Adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 4

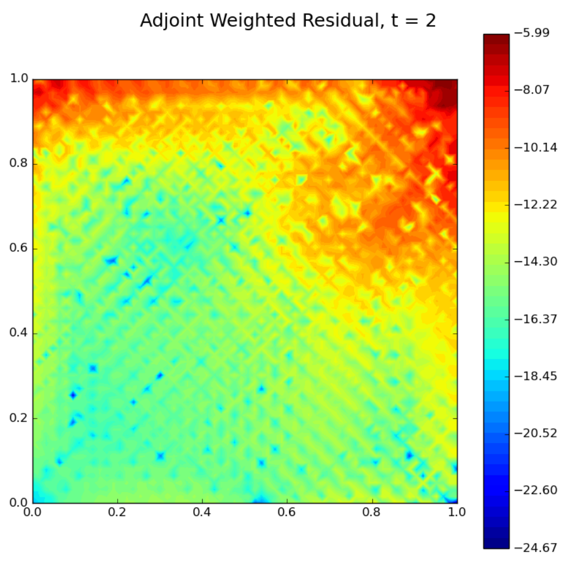

Figure C.329: Adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 2

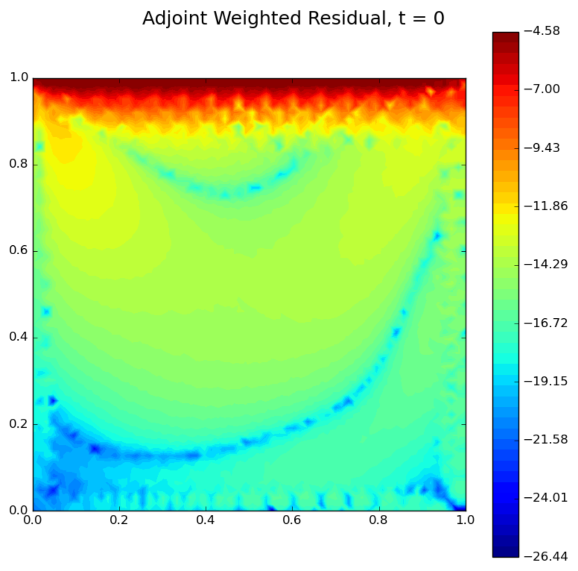

Figure C.330: Adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 0

C.5.2. POD

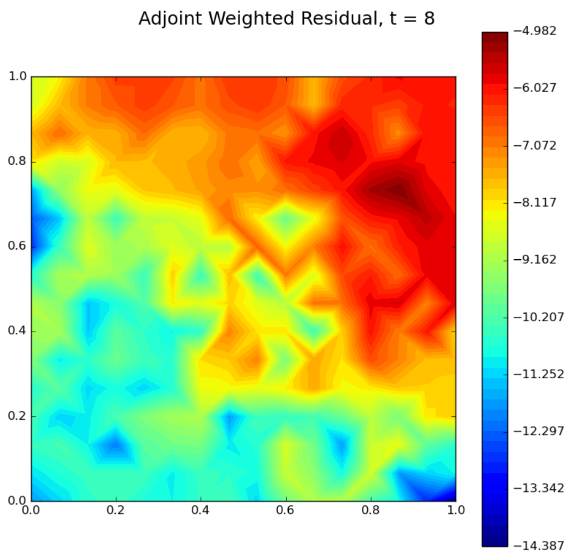

Figure C.331: POD adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 8

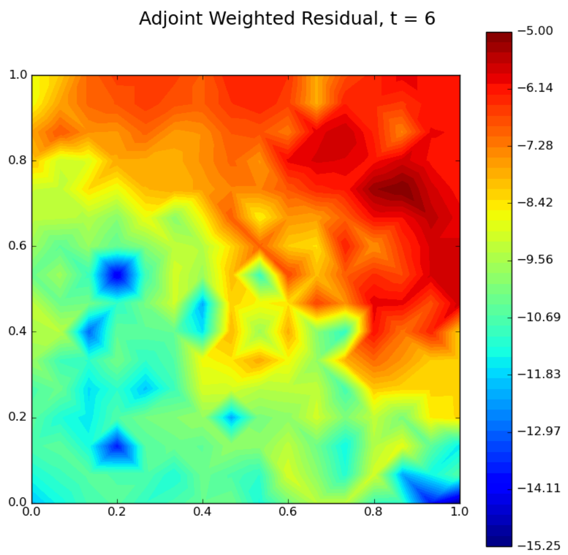

Figure C.332: POD adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 6

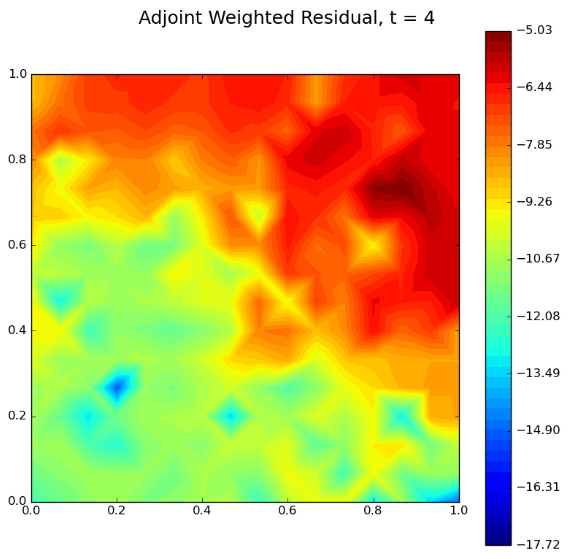

Figure C.333: POD adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 4

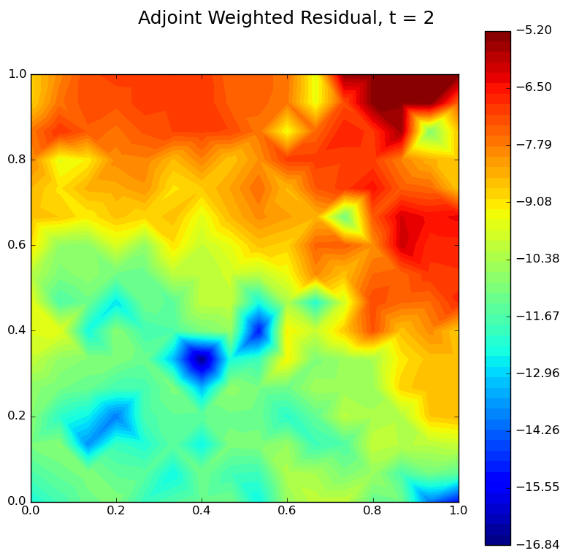

Figure C.334: POD adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 2

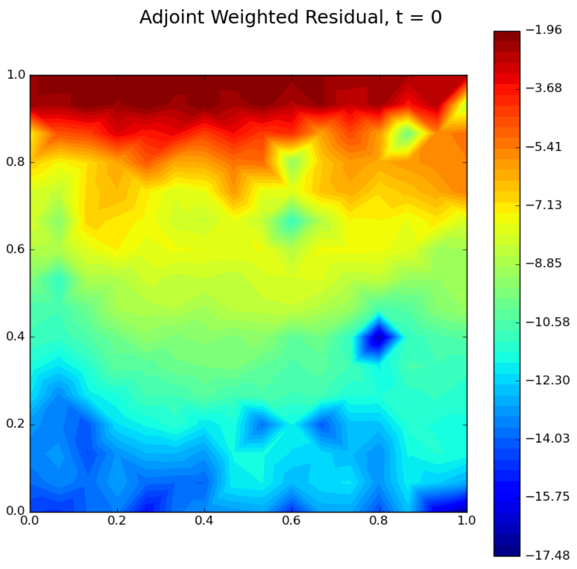

Figure C.335: POD adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 0

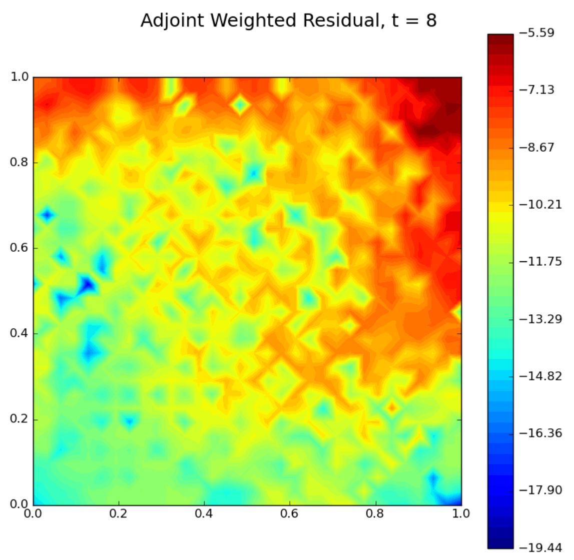

Figure C.336: POD adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 8

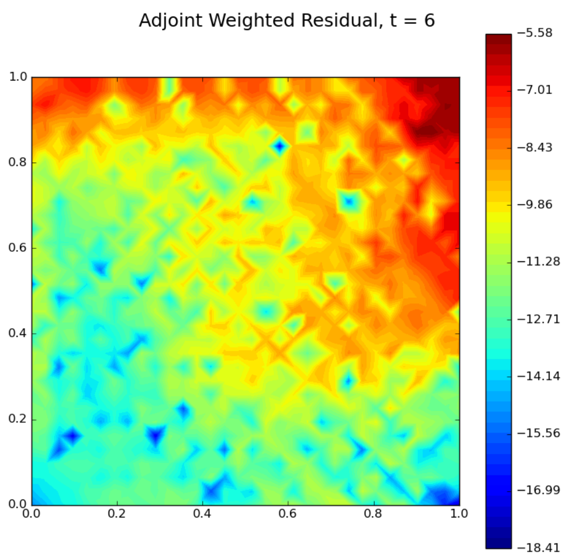

Figure C.337: POD adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 6

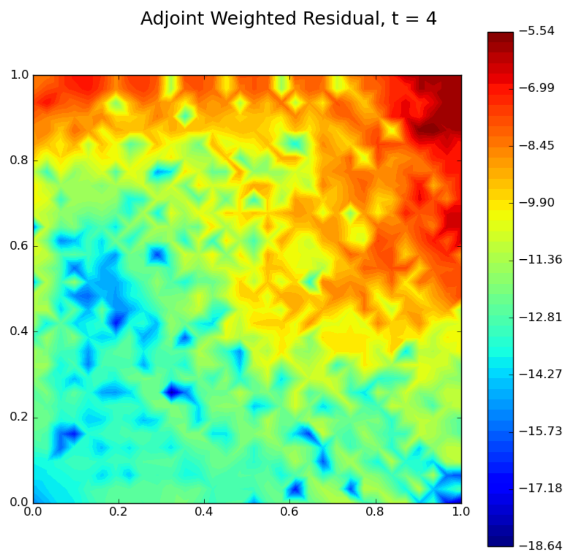

Figure C.338: POD adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 4

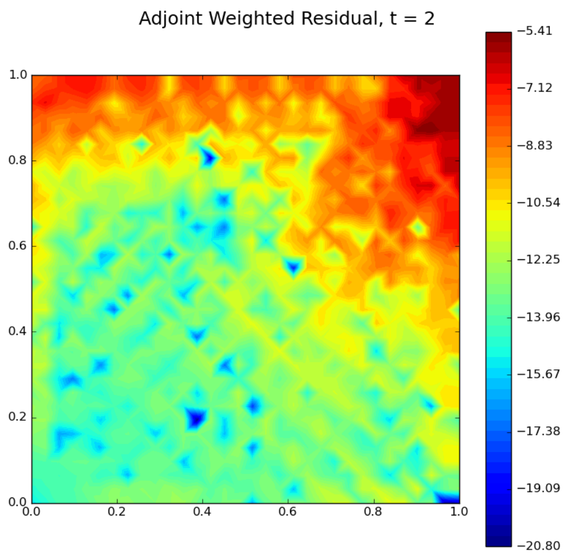

Figure C.339: POD adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 2

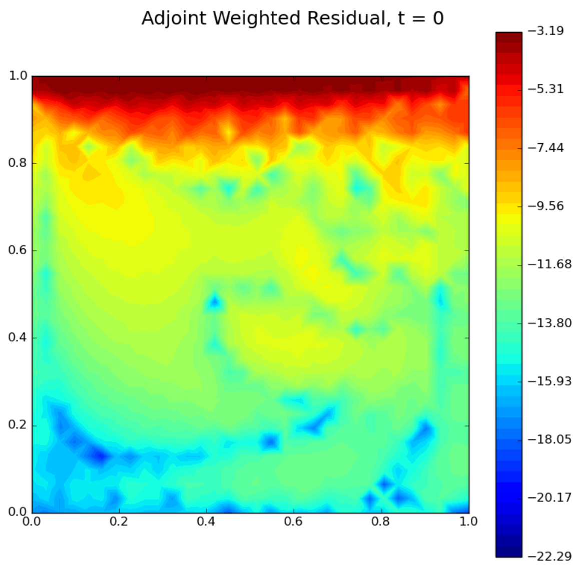

Figure C.340: POD adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 0

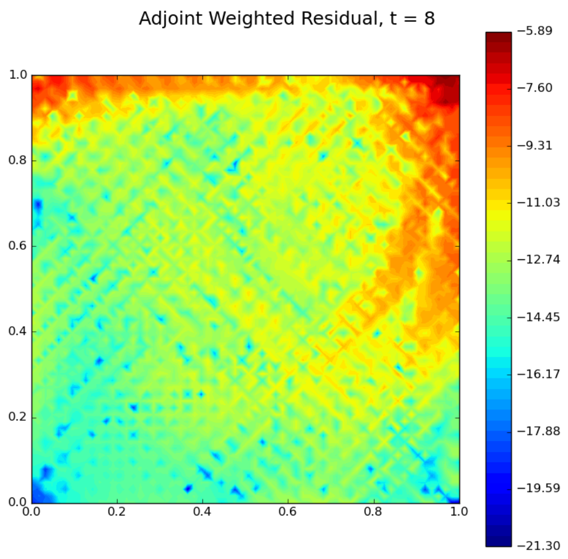

Figure C.341: POD adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 8

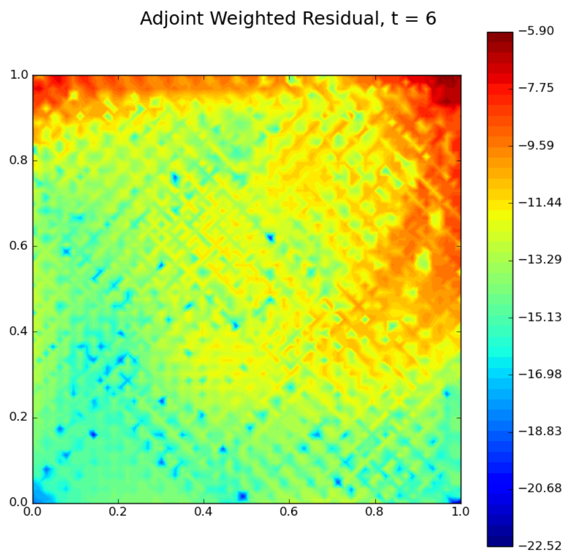

Figure C.342: POD adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 6

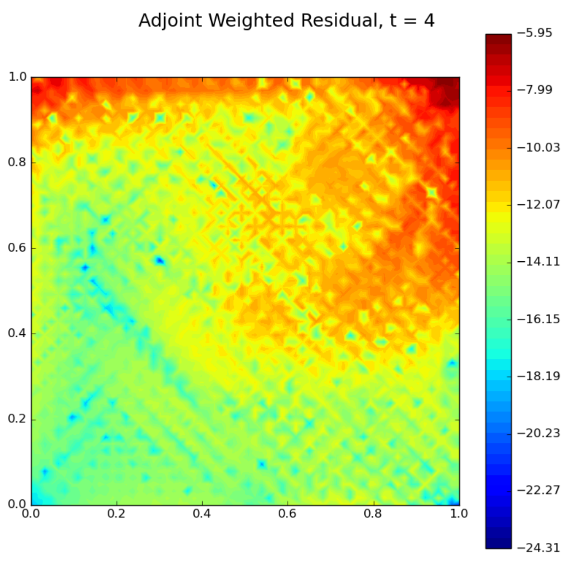

Figure C.343: POD adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 4

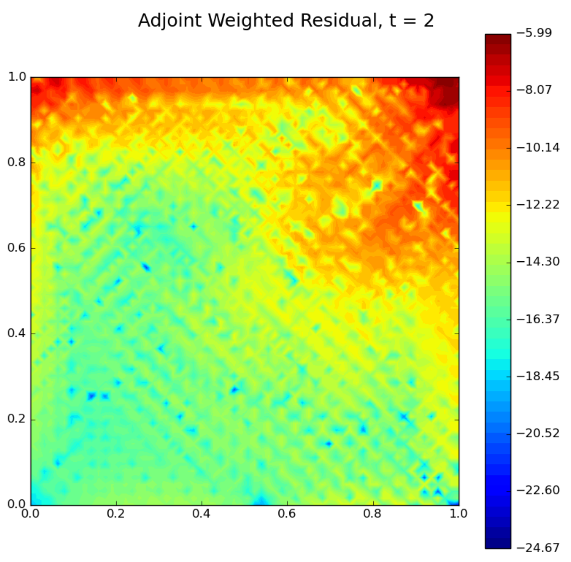

Figure C.344: POD adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 2

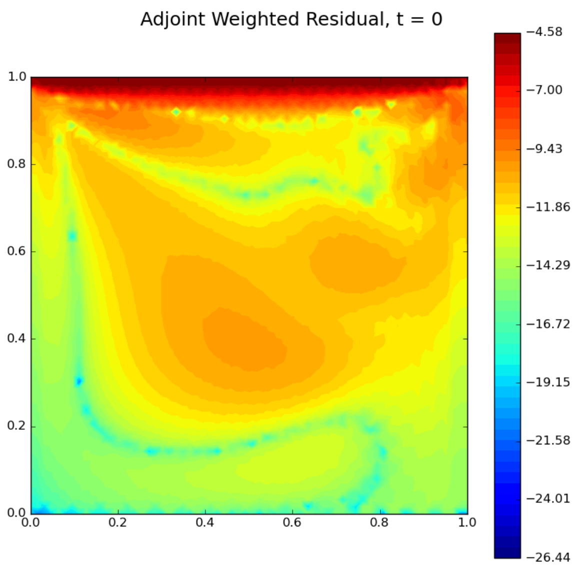

Figure C.345: POD adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 0

C.5.3. Autoencoder

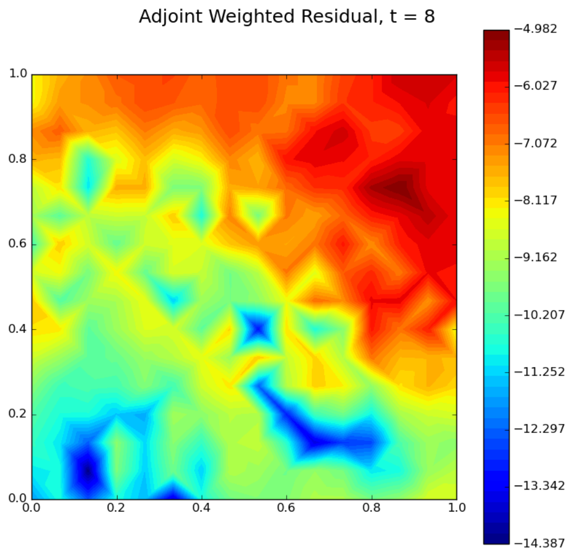

Figure C.346: Autoencoder adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 8

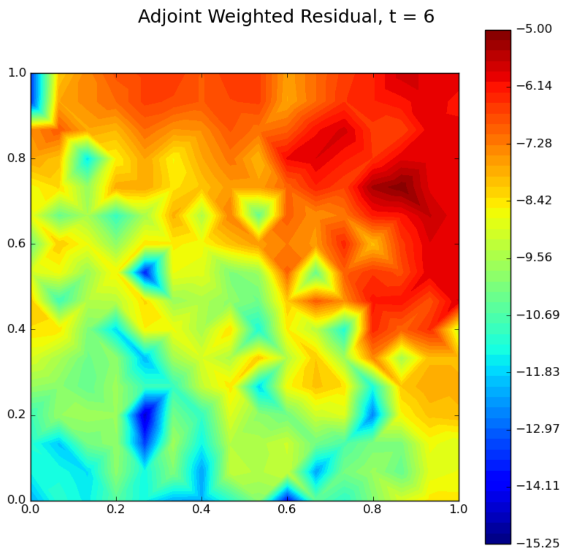

Figure C.347: Autoencoder adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 6

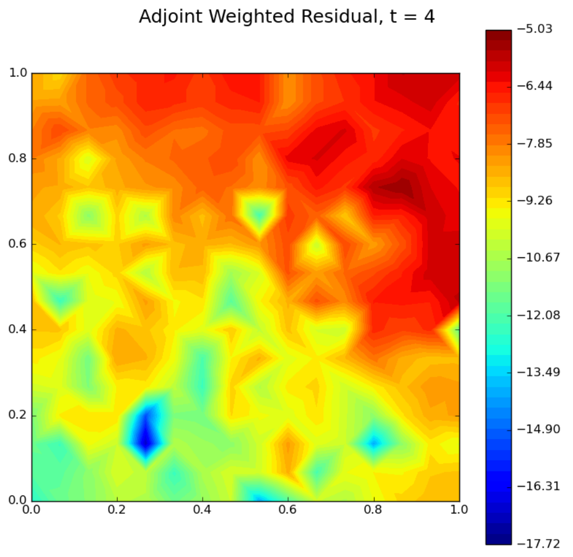

Figure C.348: Autoencoder adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 4

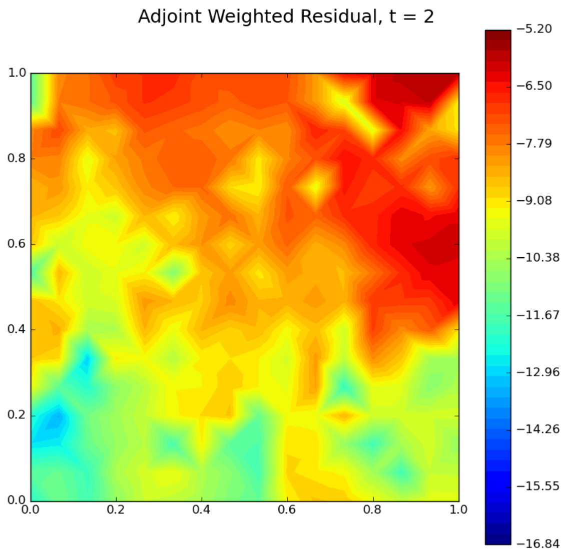

Figure C.349: Autoencoder adjoint weighted residual field for $N_h = 16 \times 16$ and Re = 500 for t = 2

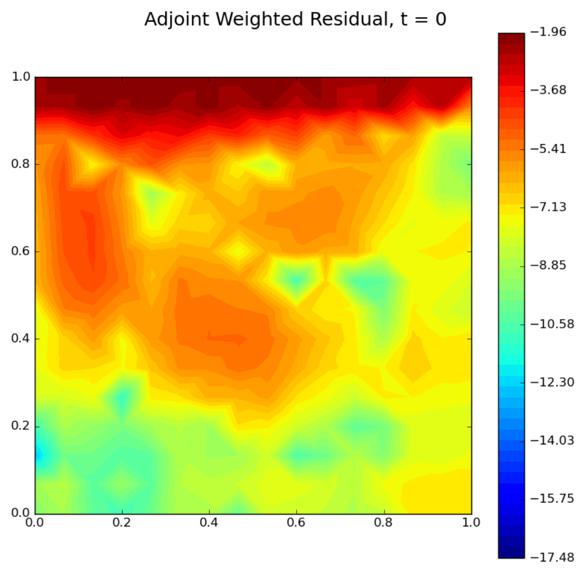

Figure C.350: Autoencoder adjoint weighted residual field for $N_h = 16\times16$ and Re = 500 for t = 0

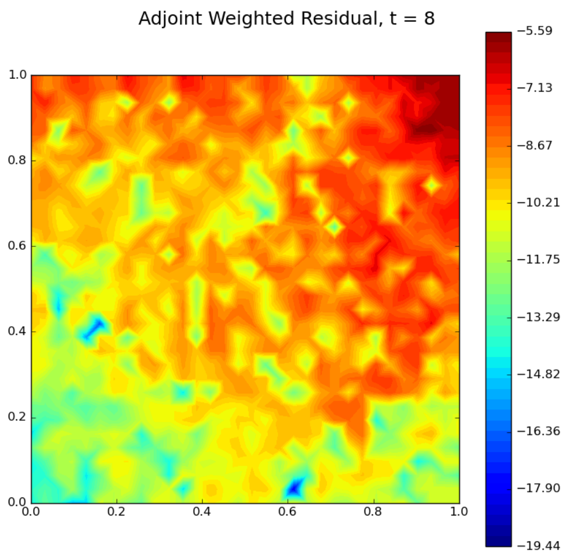

Figure C.351: Autoencoder adjoint weighted residual field for $N_h = 32\times32$ and Re = 500 for t = 8

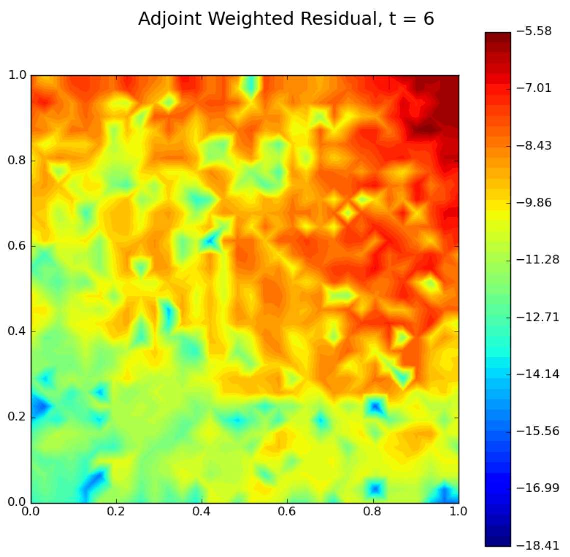

Figure C.352: Autoencoder adjoint weighted residual field for $N_h = 32\times32$ and Re = 500 for t = 6

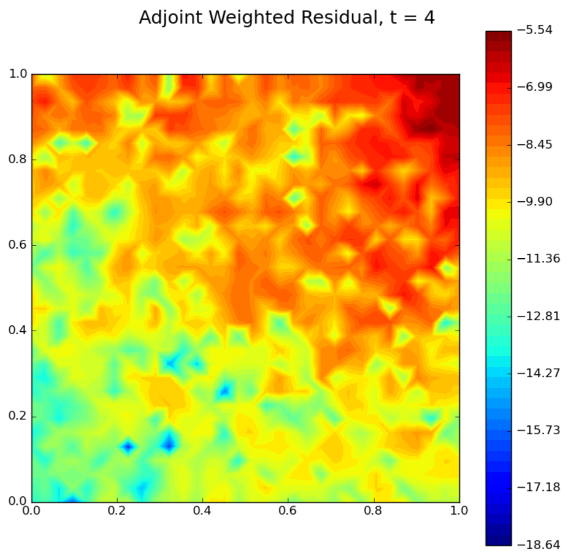

Figure C.353: Autoencoder adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 4

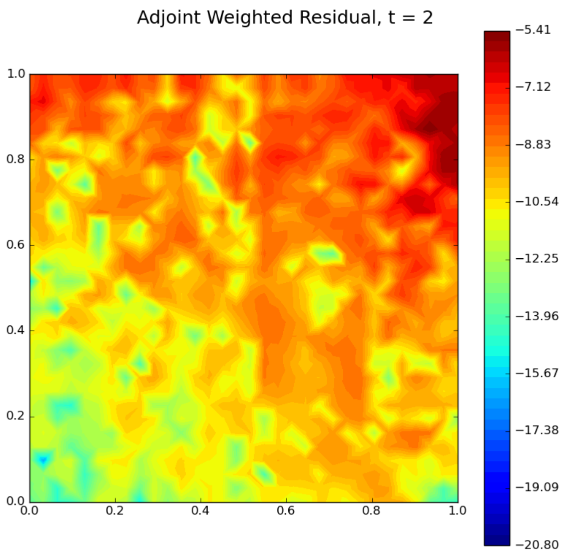

Figure C.354: Autoencoder adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 2

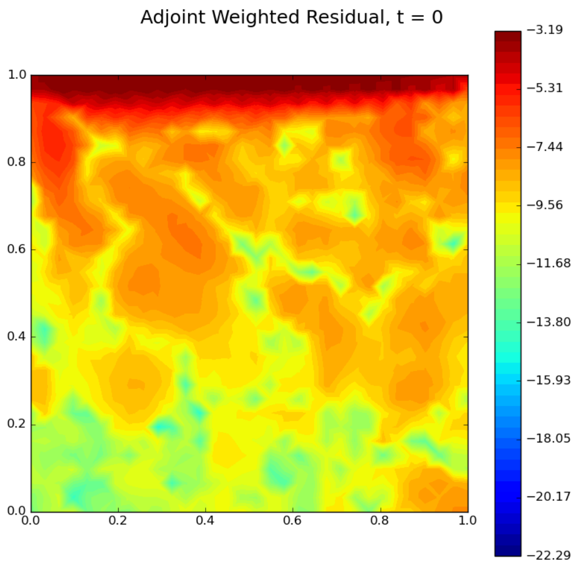

Figure C.355: Autoencoder adjoint weighted residual field for $N_h = 32 \times 32$ and Re = 500 for t = 0

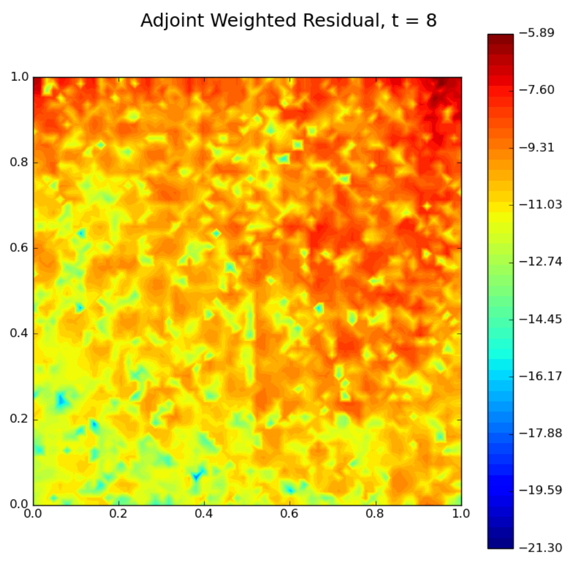

Figure C.356: Autoencoder adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 8

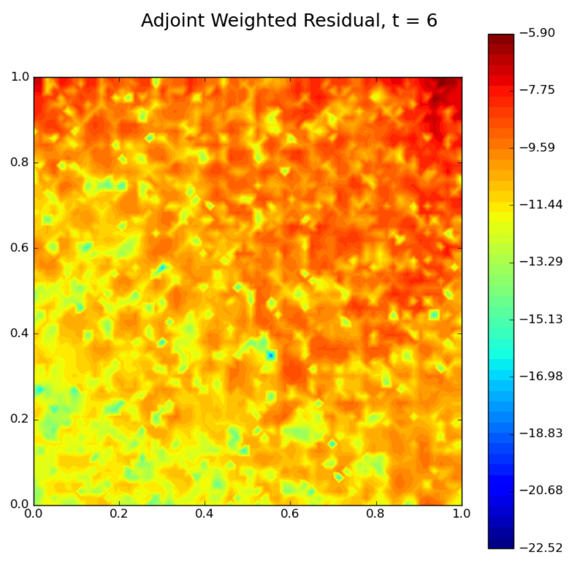

Figure C.357: Autoencoder adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 6

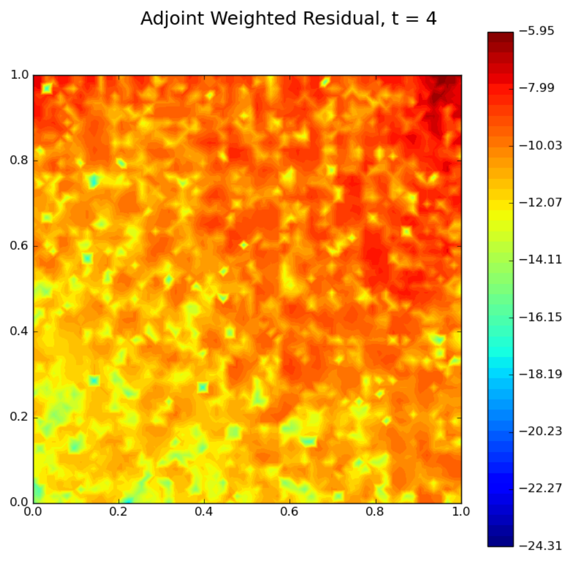

Figure C.358: Autoencoder adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 4

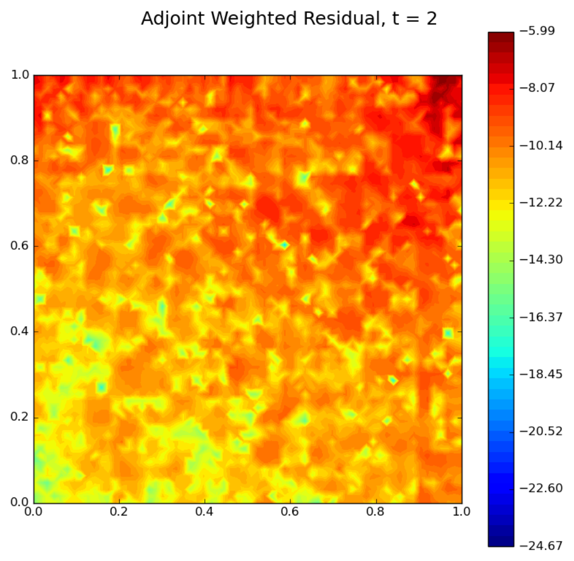

Figure C.359: Autoencoder adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 2

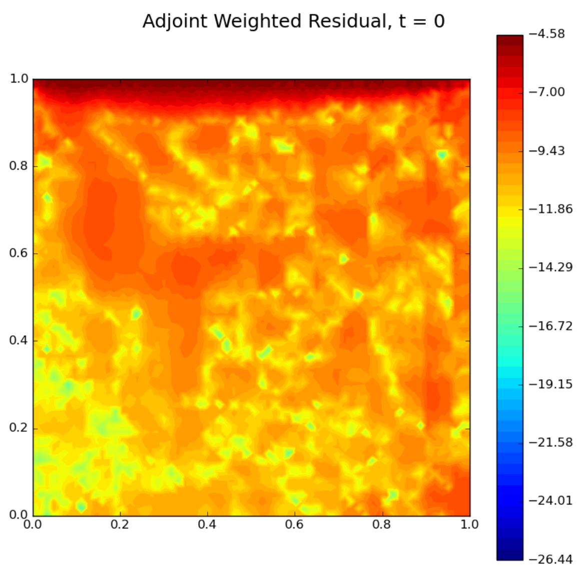

Figure C.360: Autoencoder adjoint weighted residual field for $N_h = 64 \times 64$ and Re = 500 for t = 0

C.6. Lid-driven Cavity Flow Adjoint Weighted Residual Error

C.6.1. POD

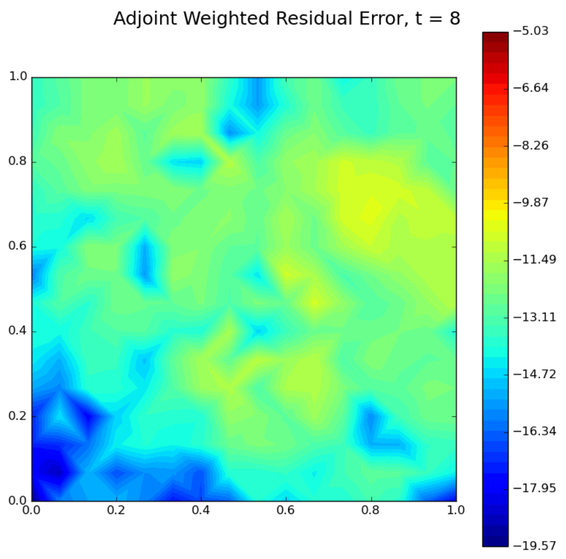

Figure C.361: POD adjoint weighted residual error field for $N_h = 16 \times 16$ and Re = 500 for t = 8

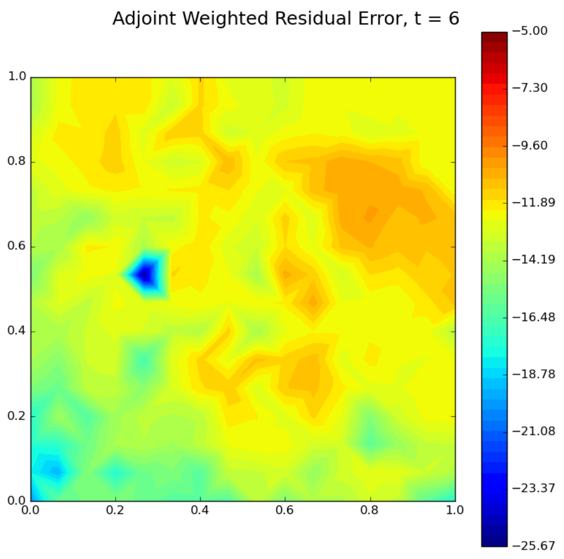

Figure C.362: POD adjoint weighted residual error field for $N_h = 16 \times 16$ and Re = 500 for t = 6

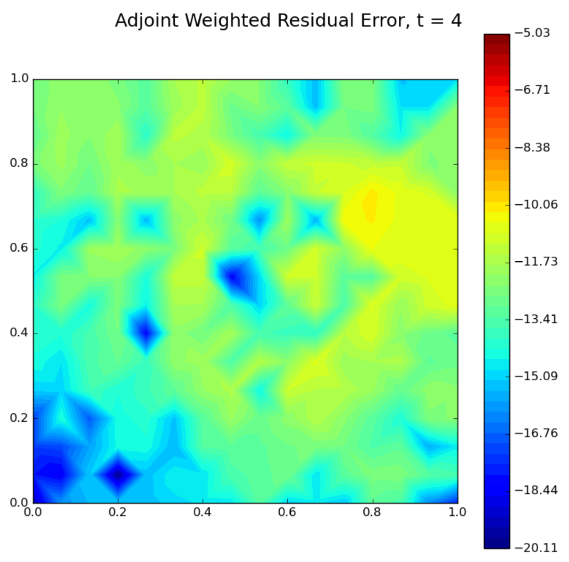

Figure C.363: POD adjoint weighted residual error field for $N_h = 16 \times 16$ and Re = 500 for t = 4

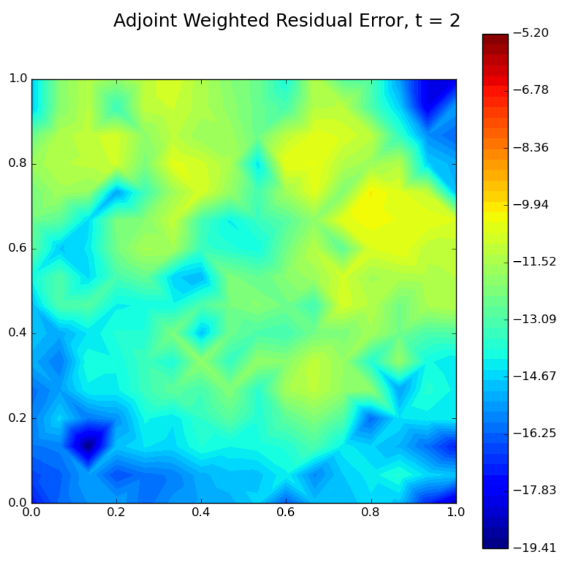

Figure C.364: POD adjoint weighted residual error field for $N_h = 16 \times 16$ and Re = 500 for t = 2

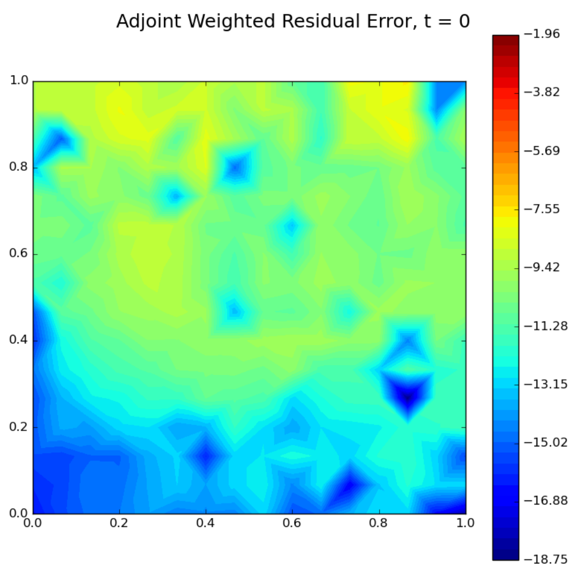

Figure C.365: POD adjoint weighted residual error field for $N_h = 16 \times 16$ and Re = 500 for t = 0

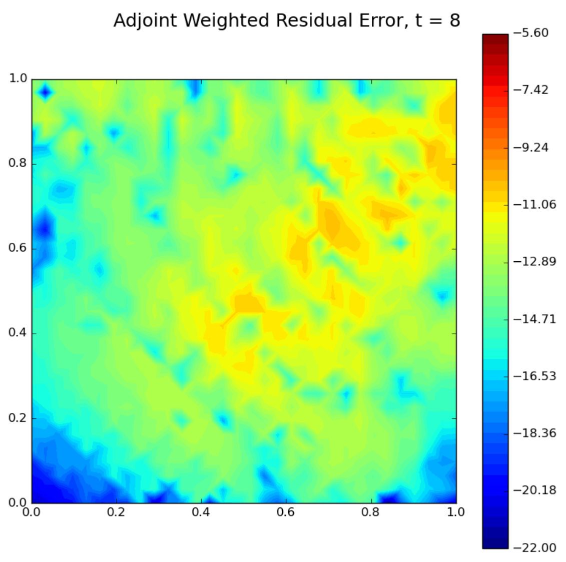

Figure C.366: POD adjoint weighted residual error field for $N_h = 32 \times 32$ and Re = 500 for t = 8

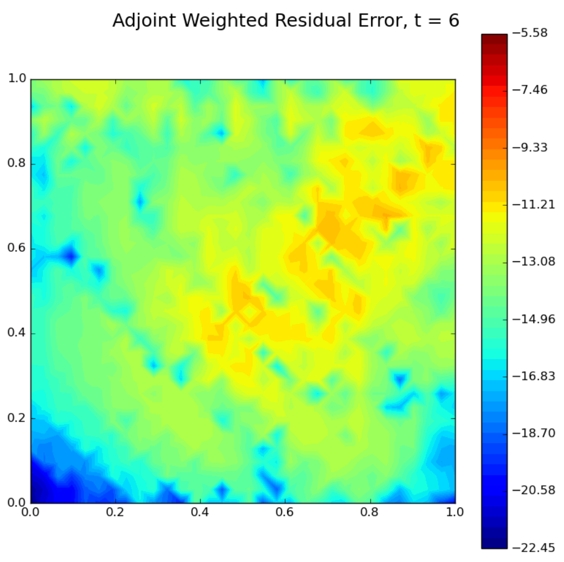

Figure C.367: POD adjoint weighted residual error field for $N_h = 32 \times 32$ and Re = 500 for t = 6

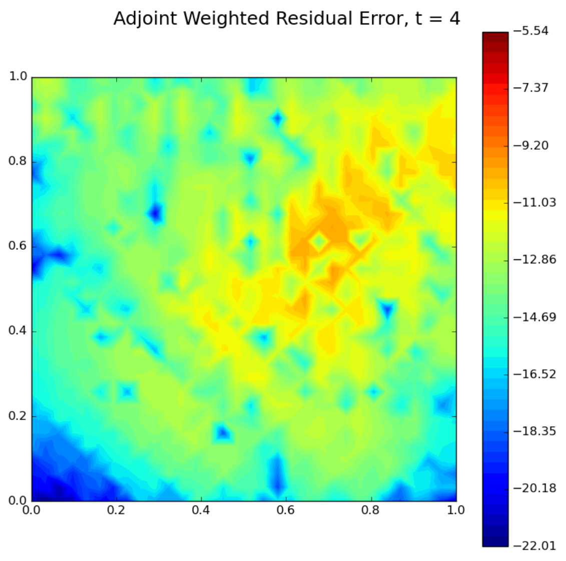

Figure C.368: POD adjoint weighted residual error field for $N_h = 32 \times 32$ and Re = 500 for t = 4

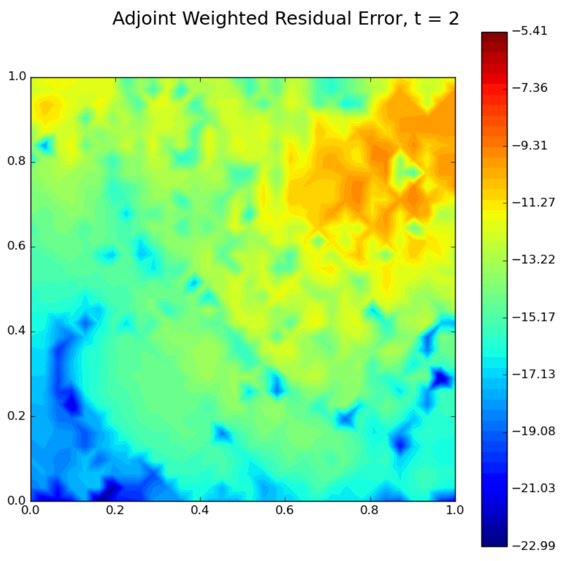

Figure C.369: POD adjoint weighted residual error field for $N_h = 32 \times 32$ and Re = 500 for t = 2

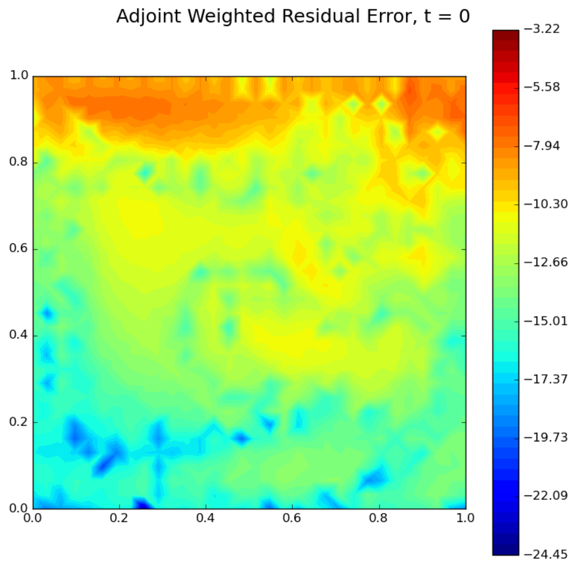

Figure C.370: POD adjoint weighted residual error field for $N_h = 32 \times 32$ and Re = 500 for t = 0

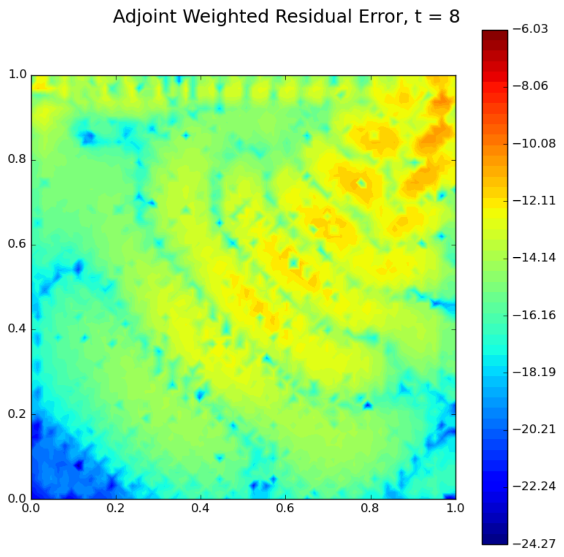

Figure C.371: POD adjoint weighted residual error field for $N_h = 64 \times 64$ and Re = 500 for t = 8

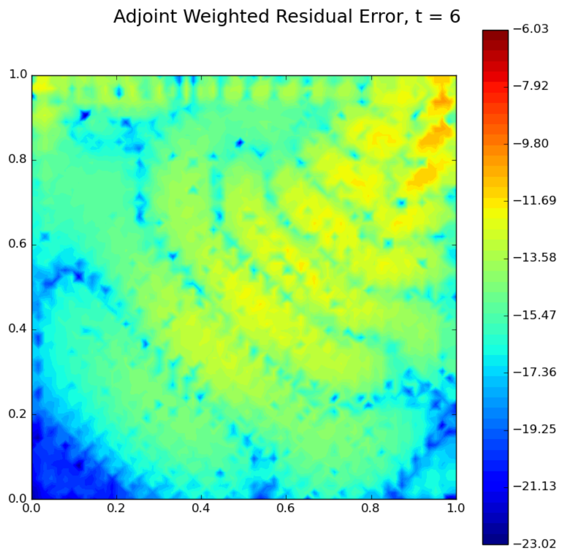

Figure C.372: POD adjoint weighted residual error field for $N_h = 64 \times 64$ and Re = 500 for t = 6

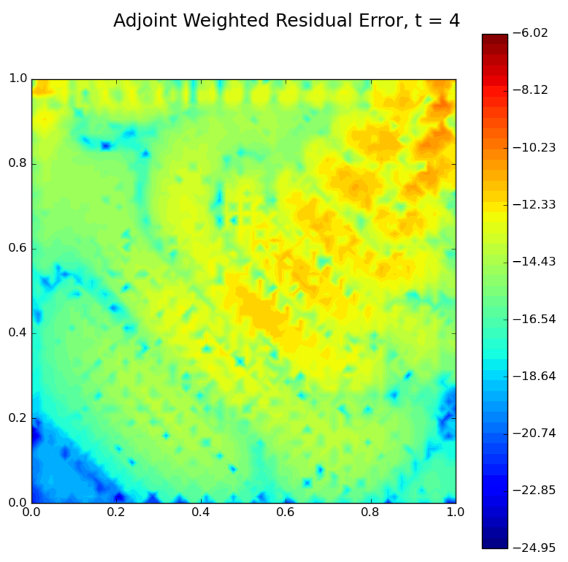

Figure C.373: POD adjoint weighted residual error field for $N_h = 64 \times 64$ and Re = 500 for t = 4

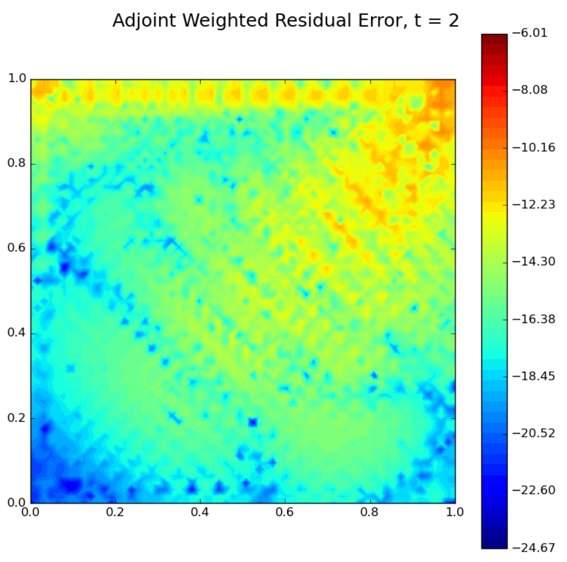

Figure C.374: POD adjoint weighted residual error field for $N_h = 64 \times 64$ and Re = 500 for t = 2

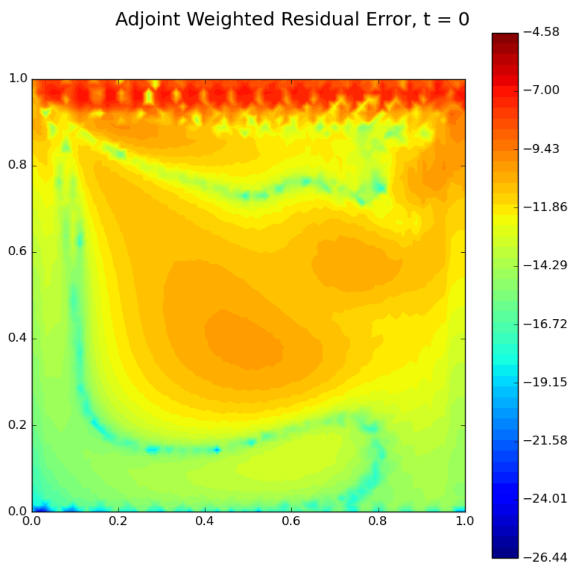

Figure C.375: POD adjoint weighted residual error field for $N_h = 64 \times 64$ and Re = 500 for t = 0

C.6.2. Autoencoder

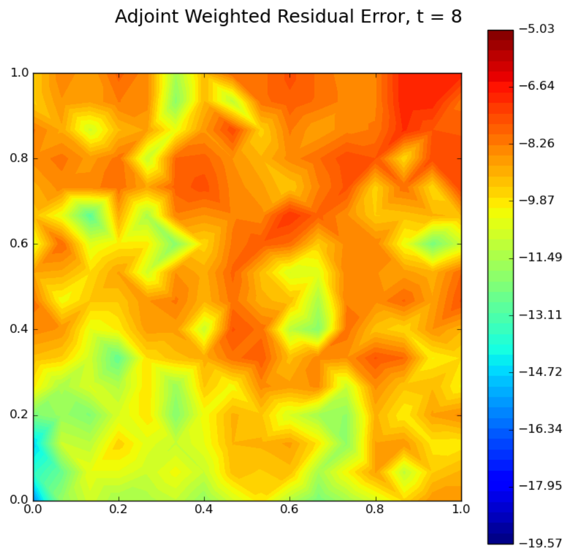

Figure C.376: Autoencoder adjoint weighted residual error field for $N_h = 16 \times 16$ and Re = 500 for t = 8

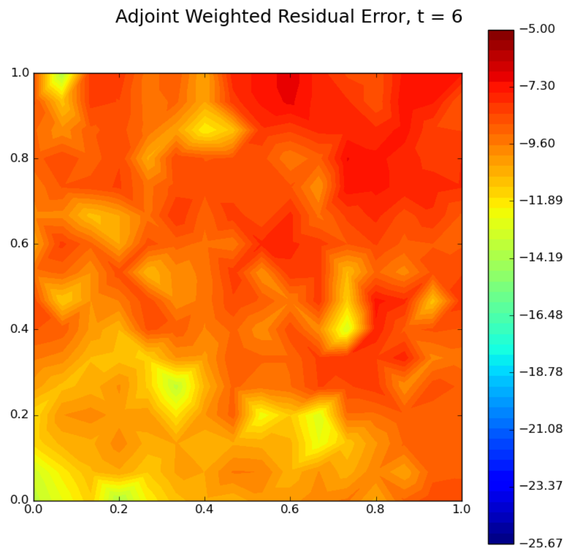

Figure C.377: Autoencoder adjoint weighted residual error field for $N_h = 16 \times 16$ and Re = 500 for t = 6

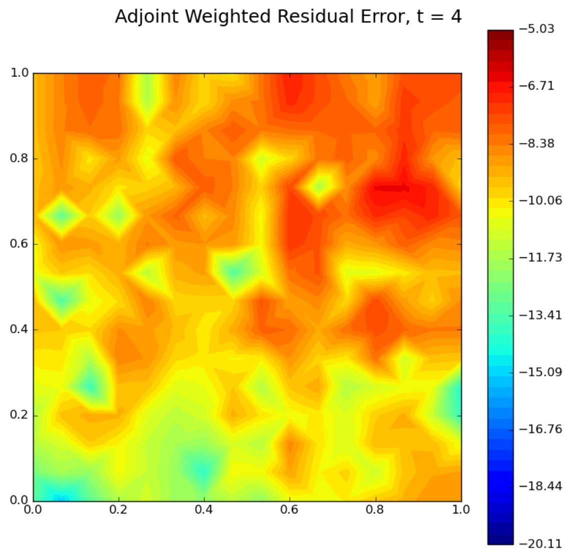

Figure C.378: Autoencoder adjoint weighted residual error field for $N_h = 16 \times 16$ and Re = 500 for t = 4

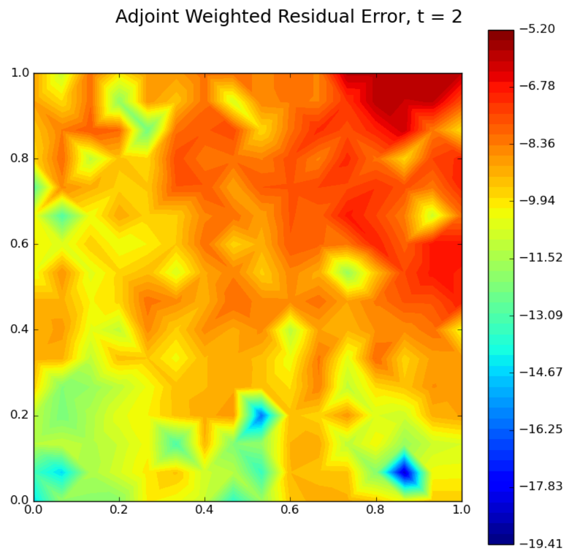

Figure C.379: Autoencoder adjoint weighted residual error field for $N_h = 16 \times 16$ and Re = 500 for t = 2

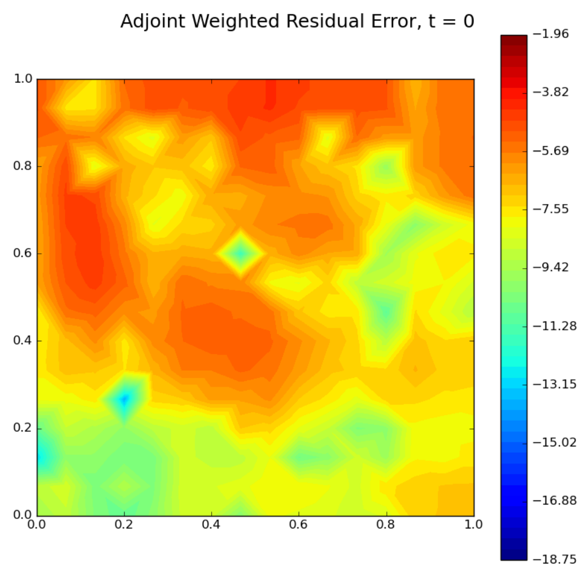

Figure C.380: Autoencoder adjoint weighted residual error field for $N_h = 16 \times 16$ and Re = 500 for t = 0

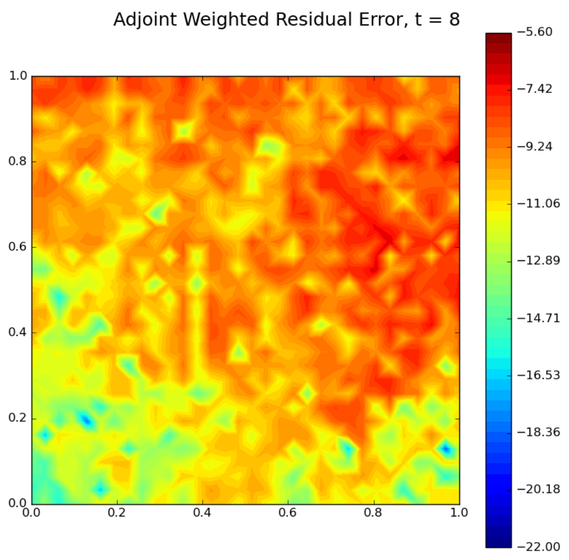

Figure C.381: Autoencoder adjoint weighted residual error field for $N_h = 32 \times 32$ and Re = 500 for t = 8

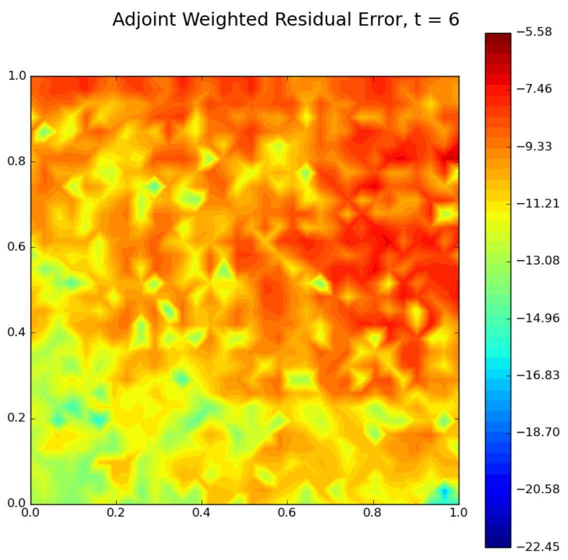

Figure C.382: Autoencoder adjoint weighted residual error field for $N_h = 32 \times 32$ and Re = 500 for t = 6

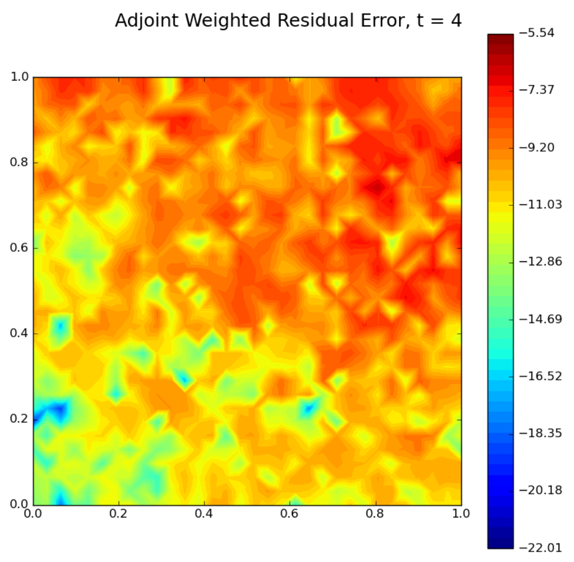

Figure C.383: Autoencoder adjoint weighted residual error field for $N_h = 32 \times 32$ and Re = 500 for t = 4

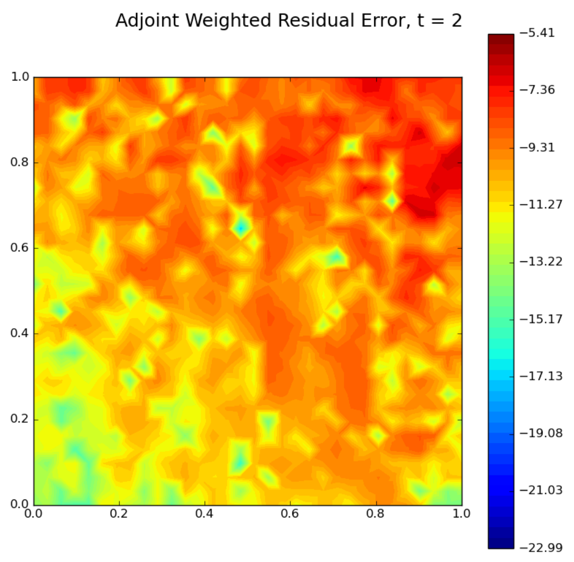

Figure C.384: Autoencoder adjoint weighted residual error field for $N_h = 32 \times 32$ and Re = 500 for t = 2

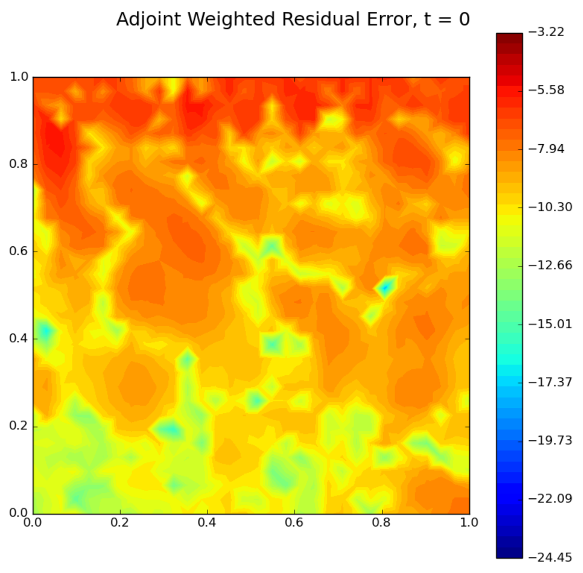

Figure C.385: Autoencoder adjoint weighted residual error field for $N_h = 32 \times 32$ and Re = 500 for t = 0

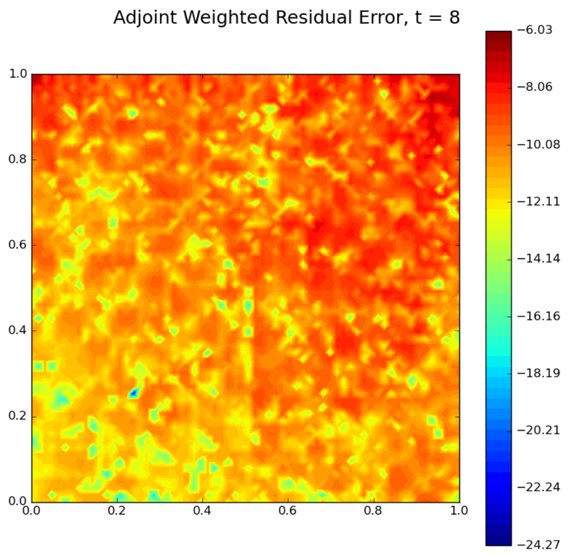

Figure C.386: Autoencoder adjoint weighted residual error field for $N_h = 64 \times 64$ and Re = 500 for t = 8

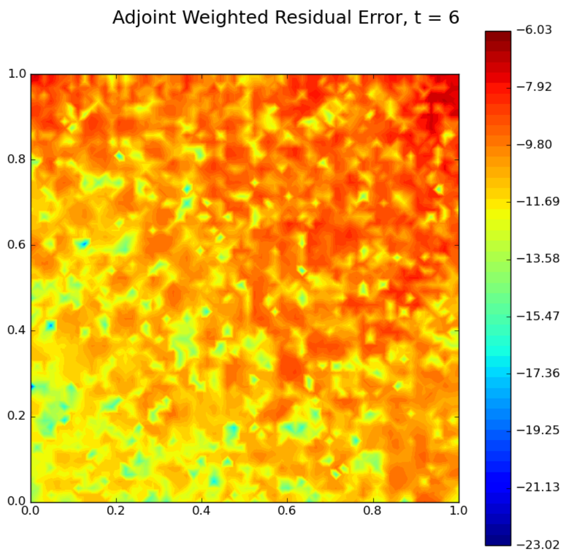

Figure C.387: Autoencoder adjoint weighted residual error field for $N_h = 64 \times 64$ and Re = 500 for t = 6

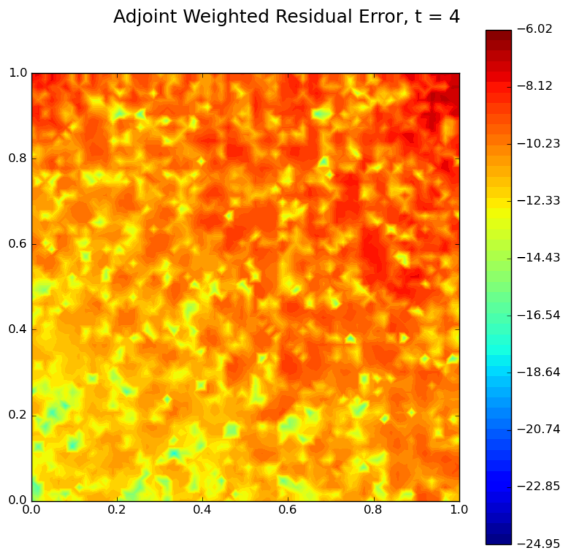

Figure C.388: Autoencoder adjoint weighted residual error field for $N_h = 64 \times 64$ and Re = 500 for t = 4

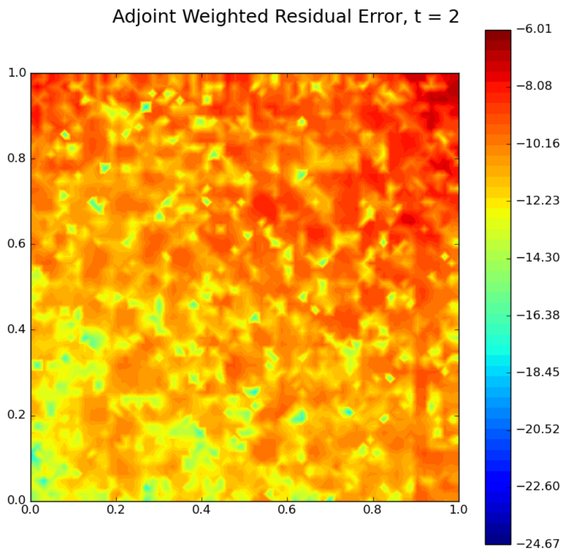

Figure C.389: Autoencoder adjoint weighted residual error field for $N_h = 64 \times 64$ and Re = 500 for t = 2

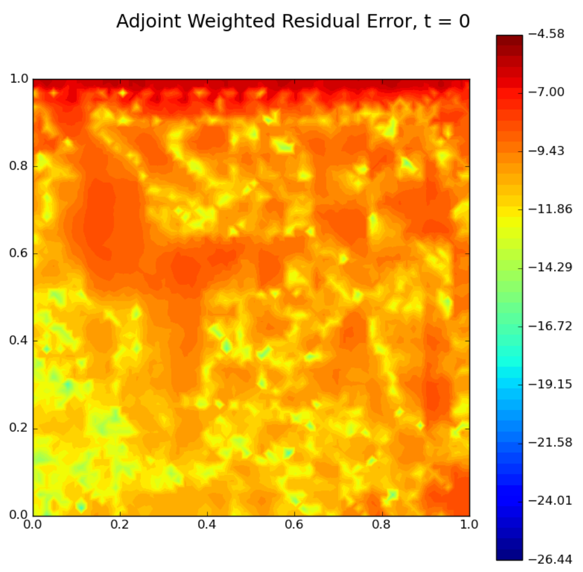

Figure C.390: Autoencoder adjoint weighted residual error field for $N_h = 64 \times 64$ and Re = 500 for t = 0