Reservoir-based prediction of convective motions

Bachelor's Project Mathematics

July 2020

Student: S.W. Kamphof

First supervisor: Dr.ir. F.W. Wubs

Second assessor: Dr. M.A. Grzegorczyk

Abstract

This thesis explores the possibility of solving Burgers' equation with a model-free reservoir predictor: an echo state network. Two types of numerical experiments are performed. One simply trains the network on spatial training data obtained through means of the finite volume method. The other trains a network on the Fourier coefficients by applying the fast Fourier transform on the spatial data. The underlying idea behind this is that due to the diffusion the higher frequency components are less relevant and could be removed from the training data, meaning one would see a decrease in computation time. The training of the Fourier coefficients performs significantly worse than the training of the spatial data when the readout matrix is reservoir focused. But both methods see an increase in performance when the readout matrix is input-focused.

Contents

Chapter 1

Introduction

In fluid dynamics, the most important equations encountered are, without a doubt, the Navier-Stokes equations. They provide a complete description of fluid flow and are used in a wide variety of applications such as the modelling of oceans or the weather. Due to the complexity of the equations, however, only few solutions are known and there are still many questions about existence and smoothness. Therefore, almost always one has to resort to numerical methods to obtain a solution. The most common approaches to obtain such a solution quite frequently rely on a discretization of the equations.

Due to the non-linear convective term, the equations display chaotic-like behaviour. Small variations can have a large impact on the final solution. To reduce the complexity of the system a bit, but conserve this term, we consider Burgers' equation. It is a simplified model of the Navier-Stokes equations that is obtained by considering the non-forced equation and dropping the pressure term. This results in the following model:

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u + \mu \Delta u, \tag{1.1}$$

where $u(x,t) : [0,L] \times (0,T] \to \mathbb{R}$ with $u(x,0) = u_0(x)$, $u(0,t) = u(L,t)$ and μ is the diffusion coefficient. This equation, like the Euler equations, can be rewritten in terms of a conservation law.

$$\frac{\partial u}{\partial t} + \nabla \cdot F(u) = 0 \tag{1.2}$$

This represents a wave equation with in this particular case, $F(\mathbf{u}) = -\frac{1}{2}\mathbf{u} \cdot \mathbf{u} + \mu \nabla \mathbf{u}$. [1]

In the simplified model the convective term is preserved and thus the system is a good testing ground for numerical approximations. The usual manner in which this set of equations is solved, is the finite volume method. A brief derivation of this method will be given in chapter 2. Spatial discretization is done through this method and a time integration such as a Runge-Kutta method is applied to obtain the approximate solution. However, often these methods require an increasingly fine grid due to the non-linearity. The convective term transfers energy into smaller scales over time, generating high wave frequency modes.[2] This can lead to aliasing, where a frequency is too high to be captured on the grid because its wave-length is smaller than the distance between two grid points

Therefore, we propose an alternate method of obtaining a solution. This method will make use of reservoir computing. A so-called reservoir is created which has certain states that depend on time. Then an input is mapped into the reservoir and combined with a state to produce a new state. This state will then be mapped to an output. During the learning phase the map from the reservoir to the output is trained so that it will map a state to the correct output. When the system has undergone sufficient learning, the output is coupled to the input to make an autonomous system that produces a solution.

This procedure shall be applied to the one-dimensional Burgers' equation which will act as a testing ground to make a prediction what the 'gain' in higher dimensions would be. Questions that will be considered are:

- *What is the complexity of the algorithm for the echo state network?*
- *Is it possible to train the network on only the Fourier coefficients within the inertial range of the solution?*
- *How many accurate time steps can be obtained for a certain amount of training steps?*

In chapter 2, an explanation of how the training data has been obtained will be given. Chapter 3 covers the explanation of how echo state networks work and its main property. This chapter also discusses the complexity of the algorithm. In chapter 4 we discuss the setup of the network and some basic examples. Then in chapter 5 we perform the actual experiments on the forced Burgers equation. Finally we will discuss the results in chapter 6.

Chapter 2

Training data

In order to obtain some data on which to train the network, we will apply the finite volume method to Burgers' equation and perform a time integration for a certain amount of steps using the four-stage Runge-Kutta method. This method is convenient since we can choose its time step in such a way that we always get stable solutions, i.e., we don't have to deal with 'numerical wiggle' in our training data.

2.1 Finite volume method

Let Burgers' equation be as defined in equation 1.2. Then bring the equation into an integral form, yielding

$$\begin{aligned} \int_V \frac{\partial u}{\partial t} dV &= \int_V (\nabla \cdot F(u)) dV \\ &= \int_S (F(u) \cdot \mathbf{n}) dS \end{aligned} \tag{2.1}$$

The second equality in equation 2.1 follows from Gauss' theorem.

Now we create a grid of N grid points in each dimension, where the distance between two adjacent grid point is h (Note that in each dimension the distance could be taken differently, which consists of equal lines/squares/cubes (1D/2D/3D). Then the volume V, the control volume, will be defined as the volume around each grid point with its boundary halfway the grid points (see figure 2.1).

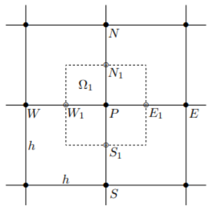

Figure 2.1: Control volume in two dimensions.[3]

In one dimension this looks like: $V = \{x \in \mathbb{R} : x_{i-1/2} \leq x \leq x_{i+1/2}\}$ where $h = \frac{L}{N}$,$x_j = \frac{j-1}{h}$,[1] $j = 1, 2, \ldots, N$ and $x_{i+1/2} = \frac{x_i + x_{i+1}}{2}$. Continuing in one dimension leads us to $u_j := u(x_j, t)$, which we can substitute into equation 1.2 and integrate.

$$h\frac{du_j}{dt} = F(u_{j+1/2}) - F(u_{j-1/2}) \tag{2.2}$$

This follows from the fact that in one dimension the boundary of V is simply the collection of the two boundary points. All that is left to do is to define the central average $u_{j+1/2} = \frac{u_j + u_{j+1}}{2}$ and define the central difference $\frac{\partial u}{\partial x}\Big|_{j+1/2} = \frac{u_{j+1} - u_j}{2}$. Substituting these approximations in (3) gives:

$$h\frac{du_j}{dt} = -\frac{1}{2}\left[\left(\frac{u_j + u_{j+1}}{2}\right)^2 - \left(\frac{u_j + u_{j-1}}{2}\right)^2\right] + \frac{\mu}{h}[u_{j+1} - 2u_j + u_{j-1}], \quad j = 1, 2, \ldots, N. \tag{2.3}$$

To deal with the grid points outside the grid we assume periodic boundaries: $u_0 = u_N$ and $u_{N+1} = u_1$.

the obtained set of equation can then be rewritten as follows:

$$\frac{d\mathbf{u}}{dt} = -\frac{1}{2h}\left[(M\mathbf{u}) \cdot (M\mathbf{u}) - (M^T\mathbf{u}) \cdot (M^T\mathbf{u})\right] + A\mathbf{u}. \tag{2.4}$$

Where $M \in \mathbb{R}^{N \times N}$ and $A \in \mathbb{R}^{N \times N}$ are defined by

$$M = \frac{1}{2}\begin{bmatrix} 1 & 1 & & & \\ & 1 & 1 & & \\ & & \ddots & \ddots & \\ & & & 1 & 1 \\ 1 & & & & 1 \end{bmatrix} \quad A = \frac{\mu}{h^2}\begin{bmatrix} -2 & 1 & & & 1 \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ 1 & & & 1 & -2 \end{bmatrix}. \tag{2.5}$$

2.2 Four-stage Runge-Kutta (RK4)

With the semi-discretization in equation 2.4 we can apply a time integration to obtain an approximate solution. In this thesis the RK4 method will be deployed:

$$\begin{aligned} \mathbf{u}_{n+1} &= \mathbf{u}_n + \frac{\Delta t}{6}(\mathbf{K_1} + 2\mathbf{K_2} + 2\mathbf{K_3} + \mathbf{K_4}), \\ \mathbf{K_1} &= f(t_n, \mathbf{u_n}), \\ \mathbf{K_2} &= f(t_n + \frac{\Delta t}{2}, \mathbf{u_n} + \frac{\Delta t}{2}\mathbf{K_1}), \\ \mathbf{K_3} &= f(t_n + \frac{\Delta t}{2}, \mathbf{u_n} + \frac{\Delta t}{2}\mathbf{K_2}), \\ \mathbf{K_4} &= f(t_{n+1}, \mathbf{u_n} + \Delta t\mathbf{K_3}). \end{aligned} \tag{2.6}$$

[1] Note that all x_j's lie in the interval $[0, L)$. When both boundary points are taken as grid points, the power spectral density will not tend to 0 for the higher frequencies, even though this is expected due to the diffusion.

Where f is the right-hand side of the discretized equation 2.4 and Δt is the size of the time step. To prevent instability in the solution, one should maintain the following stability criterium.[4]

$$4\frac{\Delta t \mu}{h^2} \leq \delta_{opt} \approx 1.6 \tag{2.7}$$

Chapter 3

Echo state networks

Reservoir computing is a branch of recurrent neural networks (RNNs). Its aim is to learn patterns from a data set for a certain period of time, $[-T, 0]$, in order to make predictions from $t = 0$ onwards. It does so by mapping an input into a 'reservoir', which is a non-linear space containing so-called states. After an input has been fed into the reservoir it is combined with a state and a non-linear input function is applied to produce a new state. The states are then collected in a matrix and a linear (feed-forward) readout is trained to map the states to the training output. This is the general approach on reservoir computing, but there are a few branches. In this thesis we consider echo state networks (ESN).

3.1 Training W_{out}

For a given amount of training inputs $\mathbf{u}(n) \in \mathbb{R}^{N_u}$, a corresponding output $\mathbf{y}(n) \in \mathbb{R}^{N_y}$ is known. Here n indicates the discrete time and it is assumed that it ranges from $-T$ to 0 with a time step of dt, so $n = 1, 2, \dots, T_d = \frac{T}{dt}$.

Our network is centered around the idea of a reservoir. This reservoir contains the states $\mathbf{x}(n) \in \mathbb{R}^{N_x}$ which depend on time and are assumed to have memory, i.e., they depend on the previous state. A current input is fed forward into the reservoir through the weight matrix $W_{in} \in \mathbb{R}^{N_x \times N_u}$ which acts as an 'Input-to-reservoir coupler'[5]. The previously known output is fed back into the reservoir through the weight matrix $W_{ofb} \in \mathbb{R}^{N_x \times N_y}$ which acts as an 'output-to-reservoir coupler'. The states also have internal connections which are described by the weight matrix $W \in \mathbb{R}^{N_x \times N_x}$. All these connections are combined together with a *bias term*[1], $\xi\mathbf{1}$, to update the states:

$$\mathbf{x}(n+1) = f\left(W_{in}\mathbf{u}(n+1) + W\mathbf{x}(n) + W_{ofb}\mathbf{y}(n) + \xi\mathbf{1}\right) \tag{3.1}$$

The function f is the state unit's non-linear output function that is applied element-wise. This function turns the reservoir into a rich and diverse dynamical system. Throughout this thesis we will use $f = \tanh(\cdot)$. Based on the current input $\mathbf{u}(n+1)$ and the state $\mathbf{x}(n+1)$, one would like to predict the output $\mathbf{y}(n+1)$. This will be the done through the readout matrix $W_{out} \in \mathbb{R}^{N_y \times (N_x + N_u)}$ and the output unit's output function f_{out}. f_{out} usually takes the form of the identity or tanh. Thus, the output is

[1]This bias term can be separately scaled for each state and can be used to manually stress importance for a certain component.

$$\mathbf{y}(n+1) = f_{out}(W_{out}\left[\mathbf{x}(n+1); \mathbf{u}(n+1)\right]. \tag{3.2}$$

With these equations, we wish to train the readout matrix W_{out} such that it is able to map a state from the reservoir to the correct output. This is done by first iterating equation 10 T_d times and storing all states at each time step. Then the matrix X containing all the states and the matrix U containing all the inputs can be concatenated.

$$X_{ext} = \begin{bmatrix} \mathbf{x}(1) & \dots & \mathbf{x}(T_d) \\ \mathbf{u}(1) & \dots & \mathbf{u}(T_d) \end{bmatrix} \in \mathbb{R}^{(N_x+N_u)\times T_d}.$$

Now equation (11) can be extended such that it covers all time steps.

$$Y = f_{out}(W_{out}X_{ext})$$

Solving for W_{out} often results in over-fitting when trying to apply the least squares method, so it is customary to use Tikhonov regularization. In some cases taking Penrose's pseudo-inverse might be advantageous as well. The former approach gives the following expression for W_{out}:

$$W_{out} = f_{out}^{-1}(Y)X_{ext}^T\left(X_{ext}X_{ext}^T + \beta I\right)^{-1} \tag{3.3}$$

where $I \in \mathbb{R}^{N_x+N_u}$, and $\beta \geq 0$. See Algorithm 1 for a general implementation of training W_{out}. It is also useful to immediately after training apply the training data U to the readout matrix and calculate the root mean squares error (RMSE) between the the obtained prediction and the actual data Y.

$$\text{RMSE} = \frac{1}{N_y}\sum_{i=1}^{N_y}\sqrt{\frac{1}{T}\sum_{n=1}^{T}\left(y_i(n) - y_i^{target}(n)\right)^2} \tag{3.4}$$

This error can be used as a reference point to increase the performance of the network by trying to decrease it.

Once we have 'trained' our read-out matrix, we can feed the reservoir new inputs and it should give outputs based on patterns it established during the training phase.

3.2 Echo-state property

The main feature of an ESN is the echo-state property. This tells us that over time, the initial conditions of the reservoir will fade and will no longer affect the states after a suffcient amount of time has passed. After one has generated a multitude of states, one can throw away the initial states and should end up with a reservoir that is independent of its initial condition. This property is useful since it gives a certain degree of robustness to the network. Each time a new network is initialized, it will keep giving a similar prediction. In this thesis the formal definition from [6] is used.

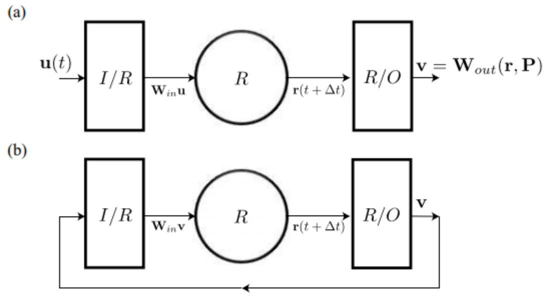
(a)
$\mathbf{u}(t)$
I/R
$\mathbf{W}_{in}\mathbf{u}$
R
$\mathbf{r}(t+\Delta t)$
R/O
$\mathbf{v} = \mathbf{W}_{out}(\mathbf{r}, \mathbf{P})$
(b)
I/R
$\mathbf{W}_{in}\mathbf{v}$
R
$\mathbf{r}(t+\Delta t)$
R/O
$\mathbf{v}$

Figure 3.1: (a) Configuration of the reservoir in the training phase. (b) Reservoir configuration in the prediction phase. This figure was taken from [5].

Definition 3.2.1. *An ESN with reservoir states $\mathbf{x}(n)$ has the echo state property if for any compact $C \subset \mathbb{R}^{N_u}$, there exists a null sequence*[2] *$(\delta_h)_{h=0,1,2,\ldots}$ such that for any input sequence $(\mathbf{u}(n))_{n=0,1,2,\ldots} \subset C$ it holds that $||\mathbf{x}(h) - \mathbf{x}'(h)|| \leq \delta_h$ for any two starting states $\mathbf{x}(0)$, $\mathbf{x}'(0)$ and $h \geq 0$.*

Despite that the general setup is called echo state network, the echo state property is not attained for arbitrary parameters. In the case of a regular ESN, this depends largely on the input, for example, for non-zero inputs $\mathbf{u}(n)$ the echo state property is often attained for $\rho(W) \geq 1$[8]. Since this requires manual tuning for which there are no formal guidelines, we will introduce a slight variant on the ESN for which there is a sufficient and necessary condition for the echo state property.

3.3 Leaky integration

A setup of an echo state network, is a so called leaky integrator. The idea being, that we can adjust the network to train on very slow dynamics by "leaking" a part of the previous state into the next. The continuous-time dynamics of a leaky integrator are described by

$$\dot{\mathbf{x}} = \frac{1}{c}(-\alpha\mathbf{x} + f(W_{in}\mathbf{u} + W\mathbf{x} + W_{ofb}\mathbf{y} + \xi\mathbf{1})), \tag{3.5}$$

$$\mathbf{y} = g(W_{out}[\mathbf{x}; \mathbf{u}]), \tag{3.6}$$

where $c > 0$ is a global time constant, $\alpha > 0$ is the reservoir neuron's leaking rate, the function f is a sigmoid function which is a bounded, differentiable real function that has a positive derivative everywhere[9] and the function g is the output activation function.

Since we only have a discrete amount of time steps available from our training data, this equation has to be discretized. In this thesis a simple Euler discretization with step-size δ is used.

$$\mathbf{x}(n+1) = \left(1 - \alpha\frac{\delta}{c}\right)\mathbf{x}(n) + \frac{\delta}{c}f\left(W_{in}\mathbf{u}((n+1)\delta) + W\mathbf{x}(n) + W_{ofb}\mathbf{y}(n)\right), \tag{3.7}$$

$$\mathbf{y}(n) = g\left(W_{out}[\mathbf{x}(n); \mathbf{u}(n\delta)]\right). \tag{3.8}$$

[2] A null sequence is a sequence that converges to 0 [7].

This provides an altered update function from equation 9:

$$\begin{aligned}\bar{\mathbf{x}}(n+1) &= \tanh\Big(W_{in}\mathbf{u}(n+1) + W\mathbf{x}(n) + W_{ofb}\mathbf{y}(n+1)\Big) + \xi\mathbf{1}, \\ \mathbf{x}(n+1) &= \frac{\delta}{c}\bar{\mathbf{x}}(n+1) + (1-\alpha\frac{\delta}{c})\mathbf{x}(n).\end{aligned} \tag{3.9}$$

By adjusting the leaking rate α, previous states will seep into the current state to prevent drastic changes in the state such that the output will match the rate at which the dynamics of the actual system change.

An advantage of using a leaky integrator ESN is that there is a sufficient and necessary condition for the echo state property.[6]

Theorem 3.3.1. *Assume a leaky integrator ESN according to equation 3.9, where the sigmoid f is the* tanh *and the output function g is bounded (for instance, it is tanh), or there are no output feedbacks, that is, $W_{ofb} = 0$. Let σ_{max} be the maximal singular value of W. Then if $|1 - \frac{\delta}{c}(\alpha - \sigma_{max})| < 1$ (where σ_{max} is the largest singular value of W), the ESN has the echo state property.*

In other words, this conditions says that the leaking rate should always be bigger than the square root of the spectral radius of $W^T W$ for the echo state property to hold. From this statement together with the fact that $\alpha\frac{\delta}{C} < 1$[3], it is also immediately observed that for the property to hold, δ and C do not require any additional restrictions. Thus they can be chosen in such a way that $\frac{\delta}{C} = 1$, eliminating them from the update function all together.

Remark. *Although this property can be useful in some cases, often when the conditions are violated the network still produces good predictions consistently. See for example the Lorenz example in Pathak's paper[5].*

[3]"A neuron should not in a single update leak more excitation than it has"[10]

Chapter 4

Setup of echo state network

4.1 Implementation of the setup

Now we can start applying echo state networks to the equation in question. The training data throughout this thesis will be obtained through means of the finite volume method and the 4-stage Runge-Kutta method as described in chapter 2. From this method, one should obtain two matrices: a matrix containing solution $\mathbf{u}(n)$ where $n = 1 \dots T_d - 1$, denote this U, and a matrix containing solutions $\mathbf{u}$ where $n = 2 \dots T_d$, denote this Y. The matrix U will act as the collection of inputs for the network and Y will be the corresponding outputs. Namely, what we want to do is to predict a next time-step based on the previous one: $\mathbf{u}(n+1) = \mathbf{y}(n)$.

The next step is to generate the weight matrices. Note that there is no need to generate W_{obf} since $\mathbf{u}(n+1) = \mathbf{y}(n)$. This turns the update equation 3.1 into

$$\begin{aligned}\mathbf{x}(n+1) &= f\left(W_{in}\mathbf{u}(n+1) + W\mathbf{x}(n) + W_{ofb}\mathbf{u}(n+1) + \xi\mathbf{1}\right) \\ &= f\left(\left(W_{in} + W_{ofb}\right)\mathbf{u}(n+1) + W\mathbf{x}(n) + \xi\mathbf{1}\right),\end{aligned}$$

where we can treat $W_{in} + W_{ofb}$ as one weight matrix.

For the generation of W_{in} the following two approaches were used throughout this thesis.

- Full: Randomly generate a matrix where each element is drawn from a uniform distribution on the interval $[-\sigma, \sigma]$.
- Sparse: Generate 1 value between $-\sigma$ and σ in each row in a random column. A single input will then excite a state instead of every input/output.

The weight matrix W is usually just defined in a sparse manner. Since the reservoir size is often quite large for complicated tasks, we can't generate a 'full' weight matrix since it would be computationally inefficient. Also, a lower sparsity often does not benefit the performance of the network in terms of the training error[11]. In this thesis we define the sparsity of the matrix in terms of the amount of entries per row, this value will be denoted by Epr. After randomly generating the values for these entries, determine the absolute value of the maximum eigenvalue and rescale W such that it has the chosen value of ρ_{max}.

The network is now ready for use and all that is left to do is to tune the 'global parameters' which are listed in table 4.1

ρ_{max}	α	β	σ
ξ	N_x	Epr	dt

Table 4.1: Global parameters

Algorithm 1: Algorithm for training W_{out}

```
Input: $\rho_{max}$, $\beta$, $\alpha$, $N_x$, $\mathbf{x}(1)$, $W_{in}$,$\sigma$, $\xi$
Output: $W_{out}$
....................................................Initialization
1: Set and scale $U = [\mathbf{u}(1) \dots \mathbf{u}(T_d)]$
2: Set and scale $Y = [\mathbf{y}(1) \dots \mathbf{y}(T_d)]$
3: Initialize $W$, $W_{in}$
....................................................Compute states
4: for $i = 0$ to $T_d$ do
5:     $\mathbf{x}(i+1) = f(W\mathbf{x}(i) + W_{in}\mathbf{u}(i+1) + \xi\mathbf{1})$
6: end for
....................................................Compute $W_{out}$
7: $X = [\mathbf{x}(1) \dots \mathbf{x}(T_d)]$
8: $X_{ext} = [X; U]$
9: $W_{out} = f_{out}^{-1}(Y) X_{ext}^T \left( X_{ext} X_{ext}^T + \beta I \right)^{-1}$
```

4.2 Complexity of the setup

To determine whether it is possible to improve on computational time compared to other methods of solving the equation, it is useful to consider the complexity of our network. The general algorithm is described in Algorithm 1. The two processes that dominate in the network is the update equation for the states and solving for W_{out}.

The dimensions of all the matrices and vectors are given in table 1 below.

For all operations, the most basic algorithms will be considered. For example, taking an inverse is considered to be $\mathcal{O}(n^3)$ using Gauss-Jordan elimination rather than something like $\mathcal{O}(n^{2.376})$ for the Coppersmith-Winograd algorithm.

Computing the $\frac{T}{dt} = T_d$ states as in Algorithm 1, line 5-7, requires the following operations.

- Multiplication of $W\mathbf{x}(i)$: $Epr \cdot N_x$. W is generated such that there are Epr entries per row.
- Multiplication of $W_{in}\mathbf{u}(i+1)$: N_x. In this thesis, W_{in} is sparsely generated in such a way that it has one entry per row.
- Multiplication of $W_{ofb}\mathbf{y}(i)$ is not necessary as will be explained in the next chapter.
- Adding the terms together with $\xi\mathbf{1}$ and applying f: $3N_x^2$.

Since this is done T_d times, this yields a total number of $T_d(Epr + 4)N_x$. This means that there is linear relation between the time it takes to compute the states and the reservoir size.

Algorithm 2: Algorithm for predicting t_d subsequent time steps on the time interval $[0, t]$.

```
Input: Initial input $\hat{\mathbf{u}}(1)$ with corresponding state $\mathbf{x}$
Output: Predicted time-series: $\hat{\mathbf{u}}(2) \dots \hat{\mathbf{u}}(t_d)$
.....................................................Initialization
1: Scale input $\hat{\mathbf{u}}(1)$
2: while $i < t_d$ do
   .....................................................Predict time-steps
3:     $\mathbf{x} = f(W\mathbf{x} + W_{in}\hat{\mathbf{u}}(i) + \xi\mathbf{1})$
4:     $\hat{\mathbf{u}}(i+1) = f_{out}(W_{out}[\mathbf{x}; \hat{\mathbf{u}}(i)]$
5: end while
6: Unscale output $\hat{\mathbf{u}}(i)$ for $i = 2 \dots t_d$
```

For the computation of W_{out}, as described in Algorithm 1, line 10, the following operations have to be carried out.

- Multiplying X_{ext} and X_{ext}^T: $T_d(N_x + N_u)^2$
- Adding βI: $(N_u + N_x)$
- Taking the inverse: $(N_x + N_u)^3$
- Multiplying with X_{ext}^T: $T_d(N_x + N_u)^2$
- Multiplying with $f_{out}^{-1}(Y)$: $N_y(N_x + N_u)T_d$. Note that $N_y = N_u$ for our purposes.

Combined with the earlier result, this accumulates to $(N_x + N_u)(2T_d(N_x + N_u) + 1 + (N_x + N_u)^2 + T_d N_u)$. This is a cubic relation with respect to $N_{xu} = N_x + N_u$ when this value is large and a quadratic one when it is small.

For the prediction phase, as described in Algorithm 2, there are 2 general operations that have to be carried out $\frac{t}{dt} = t_d$ times: the update equation and calculating $\mathbf{y}(n+1)$. The number of operations is as follows.

- Update equation as before: $Epr \cdot N_x + 4N_x$.
- Multiplication of $W_{out} \begin{bmatrix} \mathbf{x}(n+1) \\ \mathbf{u}(n+1) \end{bmatrix}$: $N_u(N_x + N_u)$.

vector	dimension	matrix	dimension
$\mathbf{x}$	N_x	W_{in}	$N_x \times N_u$
$\mathbf{u}$	N_u	W	$N_x \times N_x$
$\mathbf{y}$	N_y	W_{ofb}	$N_x \times N_y$
$\xi\mathbf{1}$	N_x	X_{ext}	$(N_x + N_u) \times T_d$

Table 4.2: Vectors/matrices and their corresponding dimension.

- Applying f_{out}: N_u.

Together this adds up to $t_d((Epr + 4 + N_u)N_x + N_u + N_u^2)$. Just like calculating the states, this expression is linear for small N_x and quadratic for large N_x.

In big-O notation these expressions are as follows:

- iterate over states: $\mathcal{O}(T_d \cdot Epr \cdot N_x)$
- calculate W_{out}: $\mathcal{O}(T_d N_{xu}^2 + N_{xu}^3)$
- predict new outputs: $\mathcal{O}(t_d(Epr \cdot N_x + N_u^2))$

In figure 3 the afore mentioned relations can be confirmed for 'small' reservoir sizes. In the left plot a linear relation between N_x and the time it takes to compute the training states can be observed as expected. Similarly, a linear relation between N_x and the prediction time can be observed in the right plot. The middle plot shows a somewhat quadratic plot.

As mentioned before, the most generic algorithms for all the computations have been taken. Most programs will use more advanced arithmetic techniques than described here, thus one can expect better performance than described here. However, this is a good basis to determine computational efficiency since it will be an upper bound.

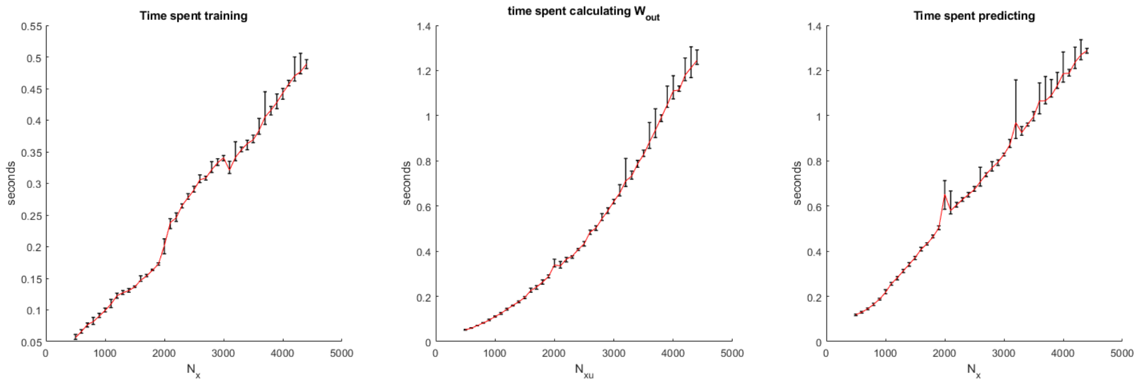

Figure 4.1: Time in seconds against the reservoir size. $T_d = t_d = 1000$. For each reservoir size 20 trials were performed. The red line goes through the median of each 20 trials and the error bars span the 1st and 3rd quartile.

Initial parameter tests

To get an idea as to how the ESN interacts with our data from Burgers' equation, two simple problems will be considered where there is not yet a forcing term involved: a single traveling wave and the diffusion of a 'random walk'. The former displays periodic behaviour over large time periods which will be a nice start for configuring the network since an ESNs main strength is replicating patterns. For the latter problem it is interesting to observe how the network deals with collecting shocks in a solution and whether it can maintain the inertial range and dissipative cutoff.

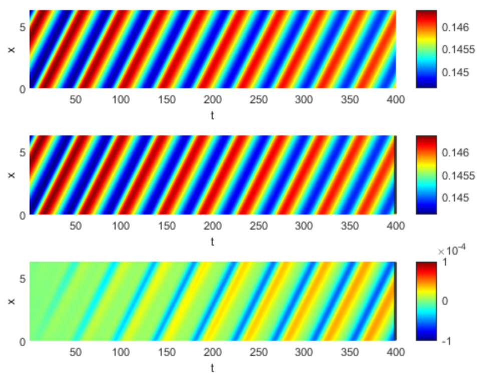

Figure 5.1: Contourplot of FVM solution (top), ESN solution (middle), difference of both solutions (bottom). Note that the first color bar corresponds to both the top and middle graph.

5.1 Single wave

For the first initial test of the echo-state network on Burgers' equation, we consider a single decaying shock on the interval $[0, 2\pi]$ as in [4]. The initial condition is given by

$$u(x,0) = \begin{cases} \exp\left(-(x-\pi)^2\right), & x \leq \pi, \\ \exp\left(-10^{-3}(x-\pi)^2\right), & x > \pi \end{cases} . \tag{5.1}$$

- The diffusion constant $\mu = 10^{-3}$.
- The number of spatial grid points is $N = 2^{11}$

- $1.6 \cdot 10^5$ time steps were applied with $\Delta t = 1.25 * 10^{-3}$, where every hundreth solution was stored as training data.

This way we can use time-steps in the ESN method of $dt = .125$ and the FVM training data still has the accuracy of a method with a time step equal to $1.25 \cdot 10^{-3}$ so that we obtain a stable solution.

With this training data, the network has been trained for 12800 time steps on the interval $[-1600, 0]$. Then from $t = 0$ onward, the output will be connected to the input in order to predict the solution for the remaining 3200 time-steps, i.e., $t \in [dt, 400]$.

The most important global parameters in the ESN have been configured as in table 5.1:

Parameter	Value	Parameter	Value
N_x	3000	β	10^{-6}
dt	0.125	EpR	150
ρ	0.6	f_{out}	id
ξ	0.3	σ	4

Table 5.1: Parameters used to obtain the results shown in figure 5.1

As predicted the network was able to pick up on the periodicity in the training data and was able to produce a solution which retained the general shape of the solution. However, from visual inspection of figure 5.1 it can be seen that the diffusion rate is not exactly matched. It appears that the network wants to flatten the solution faster than the actual FVM solution. The choice of β has a strong influence in this matter. If β is chosen as small as possible the solution matches the diffusion rate the best.

Although for small β's the readout matrix should be more accurate, it is not always possible to choose $\beta = 0$ since it is not unusual for the matrix $X_{ext}X_{ext}^T$ to be singular. In this case, there is a danger of overfitting. This is the main reason why Tikhonov regularization is widely used throughout the literature.

5.2 Random walk

Secondly, lets consider the initial velocity profile of a random walk as defined in [4].

$$u(x, 0) = \frac{1}{10}\text{Re}\left[\sum_{k=-N/2}^{N/2-1} c_k e^{ikx}\right], \quad \text{with } c_k \sim N(0, k^{-2}) \tag{5.2}$$

The training data again has $N = 2^{11}$ spatial grid points with a diffusion of $\mu = 10^{-3}$. $1.6 \cdot 10^6$ time steps have been applied where every thousandth time step has been stored. The network parameters have been chosen similarly as before.

Parameter	Value	Parameter	Value
N_x	3000	β	10^{-10}
dt	$\frac{1}{160}$	EpR	190
ρ	0.4	f_{out}	tanh
ξ	0	σ	1

Table 5.2: Parameters used to obtain the results shown in figure 5.1

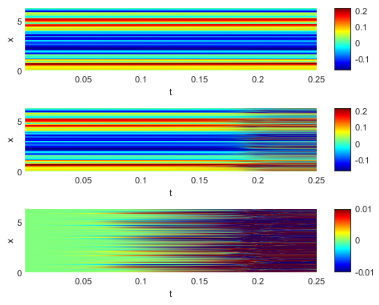

Figure 5.2: Contour plot of the 'true' solution, the predictive solution and the difference, respectively.

The network has been trained from $T = -8$ until $t = 0$. The network is expected to match the diffusion rate as in the previous case and decay at the same rate as the 'true' solution, but will probably not be able to deal with all the shocks that occur in the solution. This is confirmed by figure 5.2. After about 250 time steps, the difference between the FVM solution and the ESN prediction increases rapidly. From this figure it is not immediately clear what is actually happening. Figure 5.3 clarifies what is happening. The network has interpreted the shocks as as a sort of noise and the prediction starts oscillating around the 'true' solution. This can be properly observed in the power spectral density at 2 different time steps. The higher frequency are severely overestimated and are now dominating in the prediction. This only becomes worse over time.

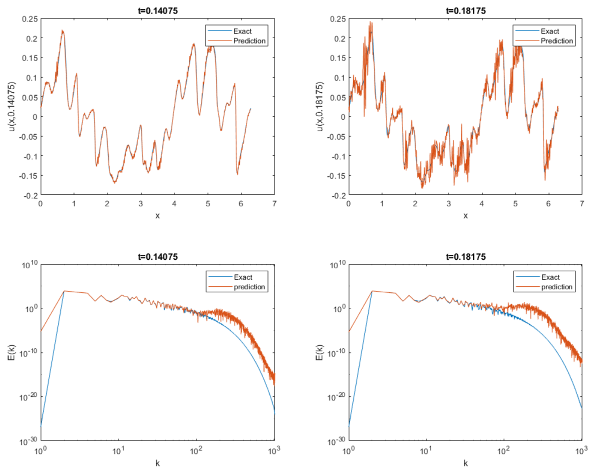

Figure 5.3: Predictions and power spectral densities at 2 arbitrary time steps.

Numerical Results

In this section we will apply echo state networks to the forced Burgers equation. By taking a specific forcing term, turbulent-like behaviour can be induced. In 3-dimensional problems turbulent behaviour is very common, therefore we would like to model similar behaviour in our test problem in the hope that we can extend the results to multiple dimensions.

In the next two sub-chapters we will consider the forced Burgers equation, $u_t = -uu_x +_{xx} +f(x,t)$, where the forcing term f is $f(x,t) = \sin(2\pi x)$. This forcing term also keeps the space average of the solution constant.

Two approaches will be applied to solve the equation. The first one will simply involve feeding the network the raw training data as obtained from our FVM method. The second method will make use of the Fourier transform. We will obtain the Fourier coefficients of the solutions at each time step and then train the network on these.

The advantage of using the second method to predict subsequent time steps is that the due to the dissipation in the solution, the higher frequencies play less of a role. You can then for example train the network on only the first 50 coefficients out of a 100 and 'throw away' the rest. This could potentially save a lot of computation time when you are working with bigger systems in more dimensions.

6.1 Raw training data

Consider the forced Burers' equation as described above with the initial condition[1] $u(x,0) = 1$. The diffusion constant is taken to several orders smaller than in the test problems: $\mu = 10^{-6}$. The domain, $[0,1)$, will be divided into $N = 300$ grid-points. Using a time step of $\Delta t = 10^{-3}$, $2 \cdot 10^5$ time steps are obtained of which the first $4 \cdot 10^4$ are used for training.

Parameter	Value	Parameter	Value
N_x	3000	β	10^{-8}
dt	0.001	EpR	8
ρ	1.6	f_{out}	tanh
ξ	0	σ	1

Table 6.1: Parameters used to obtain the results shown in figure 6.1

[1]Thus the spatial expectation of $u(x,t)$ will be 1 for all t.

A network is initialized with the parameters as described in table 6.1. From a visual inspection of figure 6.1 it appears as if the climate has been sufficiently replicated for a large portion of the time steps.

To establish when the predictions are no longer 'accurate', two errors will be defined. One that is based on the exact difference between the FVM solution and the predicted ESN solution. The first error, the NRMSE, determines for how long the prediction matches the solution 'exactly':

$$E(t) = \frac{||\mathbf{u}(t) - \mathbf{u}^{target}(t)||}{||\mathbf{u}(t)||}. \tag{6.1}$$

The norm $||\cdot||$ is the regular 2-norm, $\mathbf{u}(t)$ denotes the prediction and $\mathbf{u}^{target}(t)$ denotes the FVM solution that we are trying to predict.

The second error, the NRMSLE, will determine for how long the the prediction resembles the 'climate' of the solution. Considering that due to the chaotic-like behaviour of Burgers' equation any error will increase exponentially, the idea of a correct 'climate' is more interesting. The error to determine whether it is in the right 'climate' is given by

$$\hat{E}(t) = \frac{||\log(\hat{\mathbf{u}}(t)+1) - \log(\hat{\mathbf{u}}^{target}(t)+1)||}{||\log(\hat{\mathbf{u}}^{target}(t)+1)||}. \tag{6.2}$$

Here, $\hat{\mathbf{u}}(\mathbf{t})$ denotes the power spectral density of $\mathbf{u}(t)$. Using this error, one can also assess whether the small scales have been properly replicated, since the PSD starts decreasing exponentially for the higher frequencies. After these errors pass a certain threshold, all future predictions will be considered invalid. The threshold for $E(t)$ will be set to 0.05 and the threshold for $\hat{E}(t)$ will be set to 0.4.

In the case of the network corresponding to figure 6.1, the solution matches "exactly" for 261 time steps and the climate is successfully replicated for approximately 16275 time steps. Using a reservoir to predict an exact solution is definitely not feasible. Having used a variety of parameters shows that the network can only give an 'exact' solution (using the error criterion from equation 6.1) up to about 1200 time-steps. However, recreating the 'climate' of the solution can be done in a successful fashion. With the given example, we obtained 'accurate' time steps with a ratio of 1:3 with respect to the amount of training steps we had to take.

This solution of the network however, has an interesting feature, namely the output and the input have been scaled with a scaling constant $s = 3 \cdot 10^{-3}$. This has a huge impact on the training of W_{out}. The scaling pushes the reservoir away in the calculation of W_{out}, which causes that the inputs determine the next time step rather than the states. This seems to be the reason why the solution predicted by the ESN appears to be a steady state. Whereas the FVM solution smooths out as the higher frequency components diffuse over time, the prediction stays relatively constant.

Another initialization of a network, produces a reservoir that has more impact on the prediction of the next time step. Instead of scaling the data by 10^{-3}, the data has been shifted such that the mean of the new data is 0. The parameters that were used for this reservoir are given in table 6.2. In contrast to the network that was dominated by the output, this reservoir can produce significantly more 'exact' time steps, up to approximately 2965. However, the climate is quickly lost in the solution (after 5894 time steps). The higher frequency components do not diffuse in the reservoir but instead get amplified over time as can be seen in figure 6.3. A way to increase the duration of valid predictions for this particular problem is by recognizing that during the first

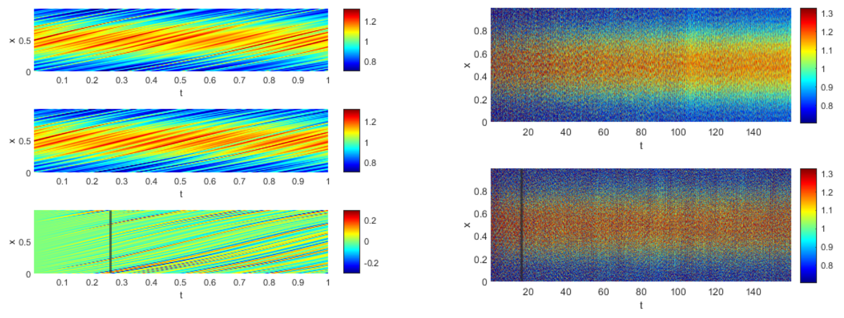

Figure 6.1: Results corresponding to table 6.1 **(left)** Contourplots on the time interval $[0, 1]$. Top panel: FVM solution. Middle panel: ESN prediction. Bottom panel: difference between the two. The gray line indicates when the $E(t) \geq 0.05$. **(right)** Countourplots on the time interval $[0, 160]$. Top panel: FVM solution. Bottom panel: ESN prediction.

10000 time steps of the solution the 'climate' is still being generated. In other words, it takes a while before the forcing term and the convection reach a sort of equilibrium before the 'climate' is constant. By discarding these time steps the network will no longer be influenced by the initial start up of the solution and it can produce more accurate time steps. This can be seen in figure 6.6 in chapter 5.3.

Parameter	Value	Parameter	Value
N_x	1000	β	10^{-4}
dt	0.001	EpR	50
ρ	0.2	f_{out}	id
ξ	0.3	σ	1

Table 6.2: Parameters used to obtain the results shown in figure 6.3

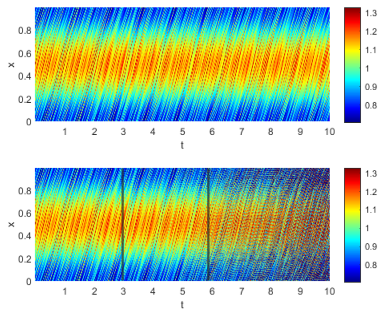

Figure 6.2: Contourplot of network where the reservoir is not pushed away. **(top)** FVM solution on the time interval $[0, 10]$. **(bottom)** ESN prediction on the time interval $[0, 10]$.

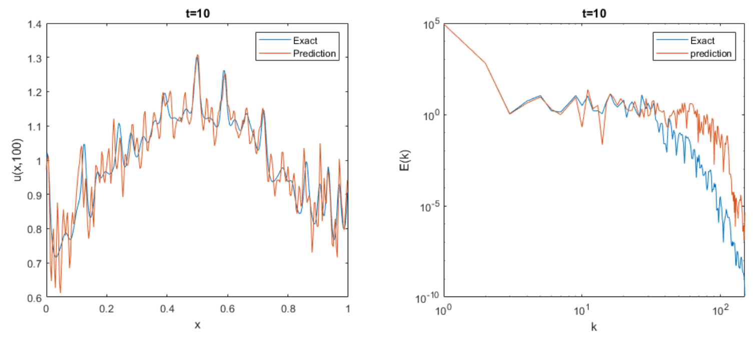

Figure 6.3: The situation at the moment that $\hat{E}(t) > 0.4$. **(left)** FVM solution and ESN prediction at t=5.894. **(right)** PSD of the FVM solution and the ESN prediction at t=5.894.

6.2 Fourier coefficients

Now the same problem will be tackled but now by using the Fourier transform on the training data and feeding the obtained coefficients to the network. As mentioned before, due to the diffusion the higher frequency components will tend to 0 and will thus have little to no impact on the overall shape of the solution. Therefore, if one can determine the inertial range[2] of the solution, it is possible to discard all the coefficients outside the inertial range. If the inertial range is small, this could potentially reduce the computational time by a lot.

We start off by training all the coefficients corresponding to the positive wave modes. These coefficients contain all the information with respect to the solution. This is because the solution is real which implies that the Fourier spectrum is conjugate even. So, we can simply train one half of the coefficients and use the symmetry to immediately obtain the other half of the coefficients as well.

The coefficients are fed into the network using the 'global parameters' as in table 6.3. Note that the coefficients can have an imaginary part, therefore the real and the imaginary part have to be split and be fed into the network independently. If one feeds the coefficients into the network as complex numbers, then the activation function f will no longer 'squash' the numbers between the values -1 and 1 but instead it will produce unbounded complex numbers[3]. So rather than training 150 inputs, we are actually training 300 inputs. So right now there is no difference in the amount of inputs compared to training the network on the raw data.

Parameter	Value	Parameter	Value
N_x	3000	β	10^{-6}
dt	0.001	EpR	18
ρ	1.8	f_{out}	tanh

Table 6.3: Parameters used to obtain the results shown in figure 6.4

[2]The Fourier spectrum of solutions of Burgers' equation follow a power law, $\mathbf{u} \propto k^{-a}$, up until the point where the diffusion takes over at the higher frequencies. The range in where the spectrum follows this power law is called the inertial range.

[3]even though the states are now complex and no longer bounded, the network can still accurately predict subsequent time steps for particular network settings. It is unclear why this is the case.

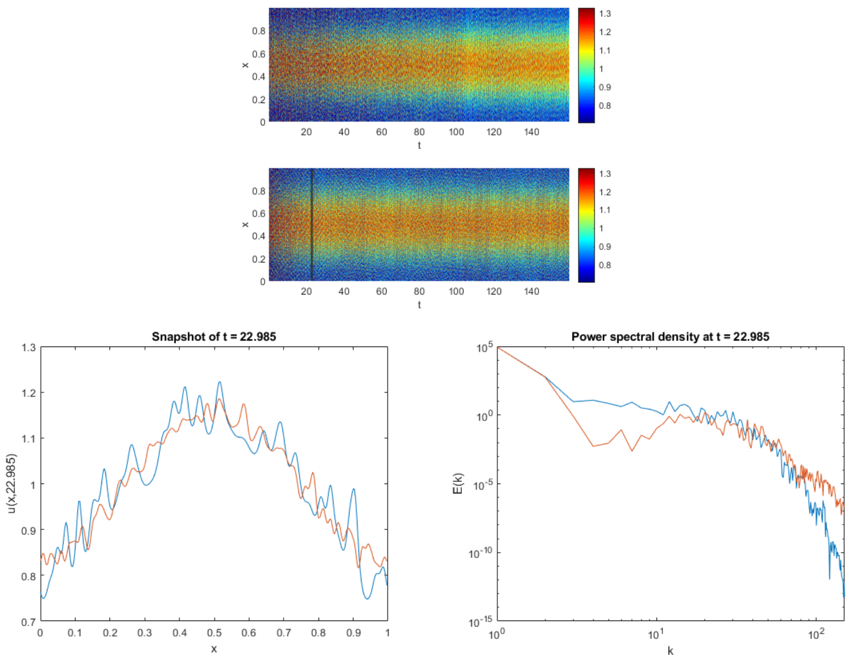

Figure 6.4: Results corresponding to table 6.3 where we train the network on the the Fourier coefficients of the positive wave modes. **(top)** Contourplot of the FVM solution and the ESN solution. The vertical line indicates at which time-step the normalized error passes the threshold. **(bottom-left)** The FVM and ESN solution of the first time-frame after the threshold has been passed. **(bottom-right)** Power spectral density of FVM and ESN solution at the first time-frame after threshold has been passed.

Again the inputs have been scaled by the scaling constant 10^{-3}. From figure 6.4 it can be observed that this gives a similar result as in figure 6.1. In chapter 6, the discussion, we shall go deeper into why these networks, which are mainly input driven, seem to perform better than those who rely more on the reservoir.

Due to the diffusion in the dynamics, the coefficients of the higher frequency modes tend to 0. They play less of a role in the overall shape of the solution. Therefore, it might be a viable option to discard them and train the network on merely the lower frequency modes. Our network will take less inputs, which means the reservoir size can be chosen smaller, increasing the computational efficiency. This time the input shall not be scaled by 10^{-3} in order to get a better idea as to how effective the reservoir itself actually is. Instead the data will again only be shifted so that its mean is 0. Figure 6.5 shows that the network produces about the same amount of valid time steps when training 30 coefficients as when all coefficients are trained. Ergo, it is definitely beneficial to train only the coefficients which are within the inertial range instead of all of them. There is however an obvious problem with this approach. Training the readout matrix where the input is not dominating gives significantly fewer valid time steps. If we compare figure 6.4 to figure 6.2, we see that we get 10 times the amount of 'exact' time steps and about 3 times the amount of time steps that resemble the correct 'climate'.

Parameter	Value	Parameter	Value
N_x	1000	β	10^{-5}
dt	0.001	EpR	50
ρ	.9	f_{out}	id
ξ	0.3	σ	0.3

Table 6.4: Parameters used to obtain the results shown in figure 6.5

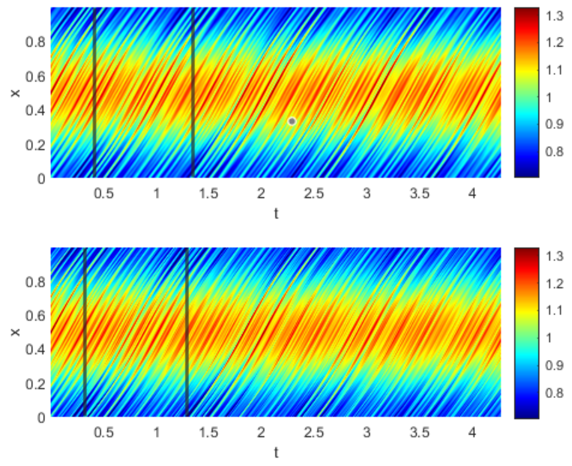

Figure 6.5: Contourplots of ESN predictions using the parameters of table 6.4. **(top)** trained on 30 coefficients. **(bottom)** trained on 150 coefficients. In for both panels the first vertical line indicates when $E(t) > 0.05$ and the second one indicates $\hat{E}(t) > 0.4$.

6.3 Comparing results

To get a clearer overview of the networks that were used in the previous sub-chapter, we iterate the networks over different reservoir size. For each reservoir size the network is reran 20 times and the median, the 1st quartile and the 3rd quartile are calculated. Figure 6.6 corresponds to the training of the spatial data with the parameters of table 6.2. Figure 6.7 & 6.8 correspond to the training of the first 150 coefficients and the first 30 coefficients respectively, with the parameters from table 6.4.

It can immediately be observed that the training of the spatial components performs significantly better even though all networks were trained on the same amount of time steps. Whereas in figure 6.6 we see that the valid time increases when the reservoir size gets larger, in the other two figures this is definitely not the case. For the 'exact' valid time, the median just shifts around the 300 a bit, and for the valid time of the climate the amount of time steps obtained becomes unstable for larger reservoir sizes. The choice of parameters does not appear to be the problem here. Rather, I expect that the problem has to do with the scaling of the data. In figure B.1 in chapter B of the appendix, the correlation and covariance matrices of the Fourier coefficients are plotted for time intervals of 10^4. Only the coefficients of adjacent wavenumbers appear to be somewhat correlated. This could be a reason why the network is unable to pick up on any patterns. Also there does not seem to be a concrete relation between the real and imaginary part. This could however be potentially resolved by training 2 separate networks, each one focusing on either the real parts or the imaginary part since those do have internal correlations.

When we take a look at the time it takes to perform Algorithm 1 (bottom left plot) and Algorithm 2 (bottom middle plot), we notice a few things. First of all, the training time for training 300 inputs is approximately the same as for 60 inputs. This is to be expected since these are relatively small reservoir sizes thus computing W_{out} does not take more than a second. The main influence comes from calculating the states. Recall that these do not depend on the size of the input since W_{in} is defined in a sparse way such that there is one entry in every row, and there are N_x rows. Ergo, no matter how many inputs we have, the training time will always be the same for a fixed N_x.

Secondly, when comparing figure 6.7 to figure 6.8 one can see that the prediction time increases by 2 times the amount when the input increases by 5 times the amount. The input size is relatively small compared to the reservoir size, since the complexity is quadratic in both components, the reservoir will dominate in the expression and overshadow N_u. In these results however we see barely any change in the quality of the solution, so it is still worth to only use 60 inputs to decrease the computational effort.

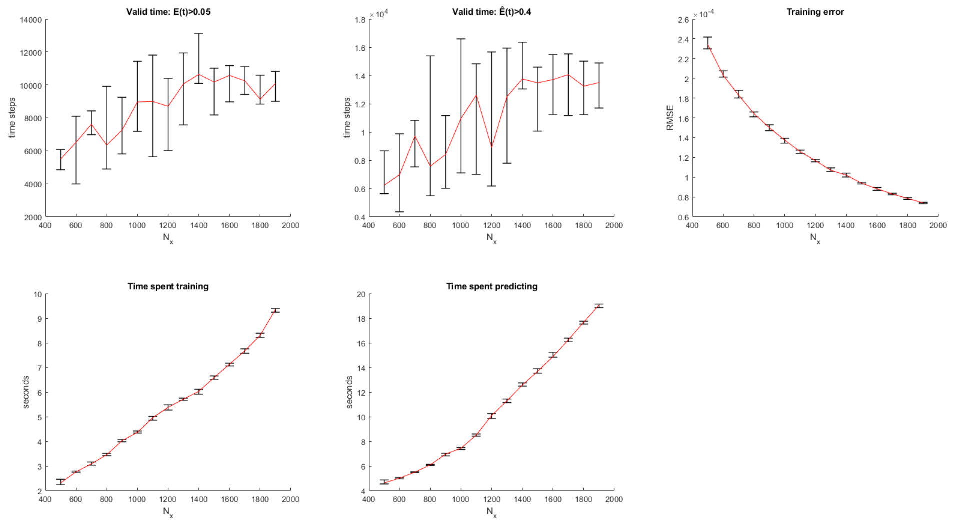

Figure 6.6: Boxplots of various statistics plotted against the reservoir size. For each N_x the 20 different networks were initialized with the parameters of table 6.2 and were trained on the interval $[-3 \cdot 10^{-4}, 0]$. The predictions were made on the interval $[0, 4 \cdot 10^4]$. The red line is drawn through the median of the statistics. The vertical lines span the 1st and 3rd quartiles.

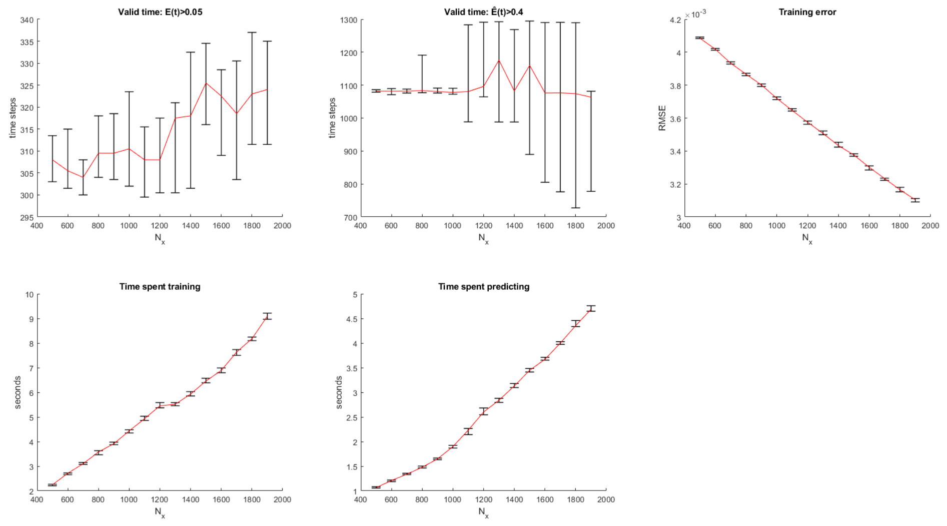

Figure 6.7: Boxplots of various statistics plotted against the reservoir size. For each N_x the 20 different networks were initialized with the parameters of table 6.4 and were trained on 150 coefficients on the interval $[-3 \cdot 10^{-4}, 0]$. The predictions were made on the interval $[0, 10^4]$. The red line is drawn through the median of the statistics. The vertical lines span the 1st and 3rd quartiles.

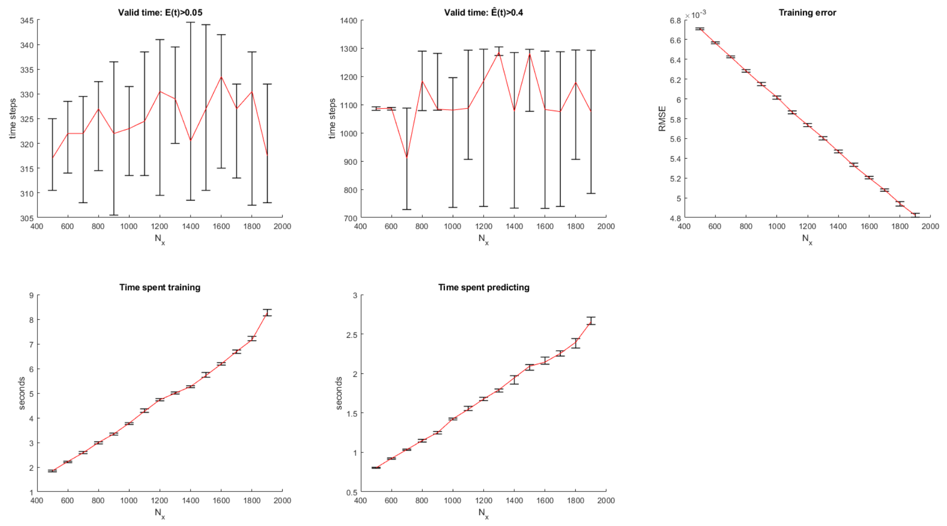

Figure 6.8: Boxplots of various statistics plotted against the reservoir size. For each N_x the 20 different networks were initialized with the parameters of table 6.4 and were trained on 30 coefficients on the interval $[-3 \cdot 10^{-4}, 0]$. The predictions were made on the interval $[0, 10^4]$. The red line is drawn through the median of the statistics. The vertical lines span the 1st and 3rd quartiles.

Chapter 7

Discussion & conclusion

In terms of accuracy, when using a mainly reservoir-focused network where the states contribute the same amount as the previous input, it was observed that training the network on spatial data gives a much better quality of prediction than when training the real and imaginary parts of the Fourier coefficients. However, when the data is scaled in such a way that in computation of W_{out} is mainly dependent on the previous on the input, we see that that both methods drastically increase their validity of the 'climate' (At the cost of the 'exact' validity). Our method has transformed into something that resembles dynamic mode decomposition (DMD)[12]. The difference here, however, is that subsequent flow fields $\mathbf{u}(n)$ and $\mathbf{u}(n+1)$ have an extra linear term in their relation: $\mathbf{u}(n+1) = A_{12}\mathbf{u}(n) + A_{11}\mathbf{x}(n+1)$. Note that $W_{out} = \begin{bmatrix} A_{11} & A_{12} \end{bmatrix}$ and that $\left(A_{11}\mathbf{x}(n+1)\right)_i << \left(A_{12}\mathbf{u}(n)\right)_i$ implying $\mathbf{u}(n+1) \approx A_{12}\mathbf{u}(n+1)$. A few things to note here are:

- Some additional experiments show that this relation does not seem to work when we discard the states all together.
- Since inputs and outputs are now significantly smaller than 1, the function $f_{out} = \tanh$ basically acts as the idenity function. The output is now just an actual linear combination of the states and the input: $\mathbf{y}(n) = \mathbf{u}(n+1) = W_{out}\begin{bmatrix}\mathbf{x}(n+1) \\ \mathbf{u}(n)\end{bmatrix}$.

Rather than scaling the input, we can also just manually choose f to be the identity. Then we can rewrite the linear readout expression as $\mathbf{u}(n+1) = A_{12}\mathbf{u}(n) + A_{11}\mathbf{x}(n+1)$. This can be considered a special form of DMD, namely DMD with control [13], where the control comes from the reservoir.

To come back to the problem of training on the Fourier coefficients, the problem might lie in the dynamics of the input. As you can see in figure A.2 in chapter A, the coefficients of different wavenumbers evolve at different speeds over time. Due to the internal connections within the states, these rates of evolution are mixed up and 'dissappear' into the new dynamics of the reservoir. Since the connections in the reservoir are random, we can not choose α such that each state matches the dynamics of one the inputs/outputs. It might be possible through means of a time-scaling constant, to line up the different dynamics in order to improve the performance of the network trained on Fourier coefficients.

In terms of time, echo state networks are rather efficient. For reservoir sizes of order 10^3 the training time has an approximate linear relation with the reservoir size. Similarly for the prediction phase there is a linear relation when N_x is small. If we want to consider higher dimensional problems, for example Navier-Stokes in 3 dimensions, then the number of inputs will increase significantly. For example if we discretize the grid in 300 grid points in each direction then we obtain $27 \cdot 10^6$ grid points in total. In order to obtain accurate predictions, N_x will have to be of a similar order. Since the complexity for large reservoir sizes is not that much of an improvement on than that of the FVM method, for example. However, we do know that for smaller input size we can take smaller reservoir size. One could divide all the grid points in different overlapping groups of n_u by n_u by n_u, where $n_u << N_u$. Then for each of these groups a network is trained to predict the time evolution. This way the computation time is kept down. This approach has already been somewhat successfully deployed for the one dimensional Kuramoto-Shivashinky equations [14] and given the success of our spatial training, it could perhaps be deployed for our purposes.

To continue this research, one could look into training different representations of the data, such as the polar coordinates of the Fouier coefficients. The radii of these can be rather successfully trained, but the problem is in the angle. Due to the fact that Matlab continues at $-\pi$ after π has been passed, this data has a lot of discontinuities which the network can not deal with. By constructing an algorithm that extends the interval $[-\pi\pi]$ to an interval big enough so that it covers all jumps, the data can be reprocessed into continuous curves such that the network is able to train on the data.

Chapter A

Fourier coefficients: time evolution

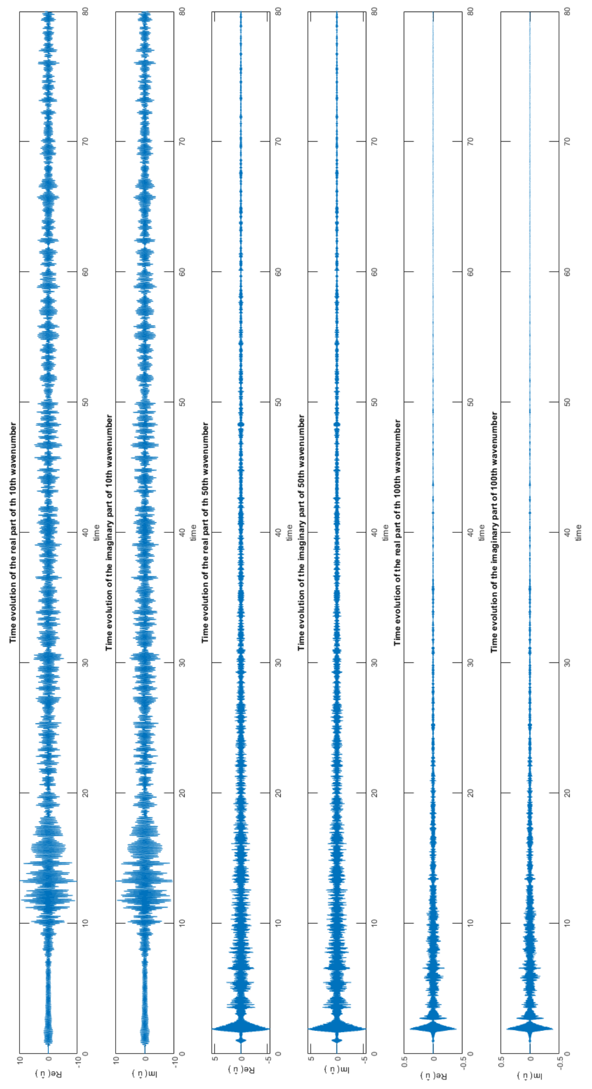

Figure A.1: Time evolution of the real and imaginary part of three Fourier coefficients on the time interval $[0, 80]$. The first two figures correspond to the 10th wavenumber. The 3rd and 4th figure correspond to the 50th wavenumber. The last two figures correspond to the 100th wavenumber. In each case the left figure is the real part and the right figure is the imaginary part.

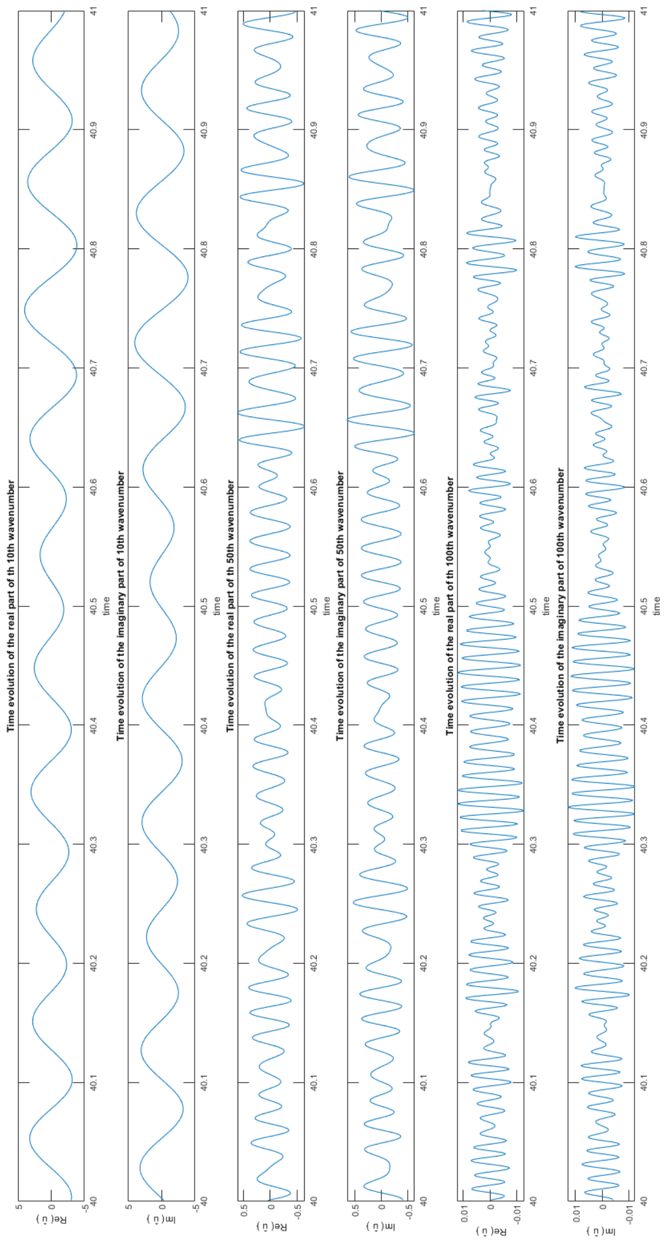

Figure A.2: The same time evolutions as in figure A.1, but zoomed in on the time interval $[40, 41]$.

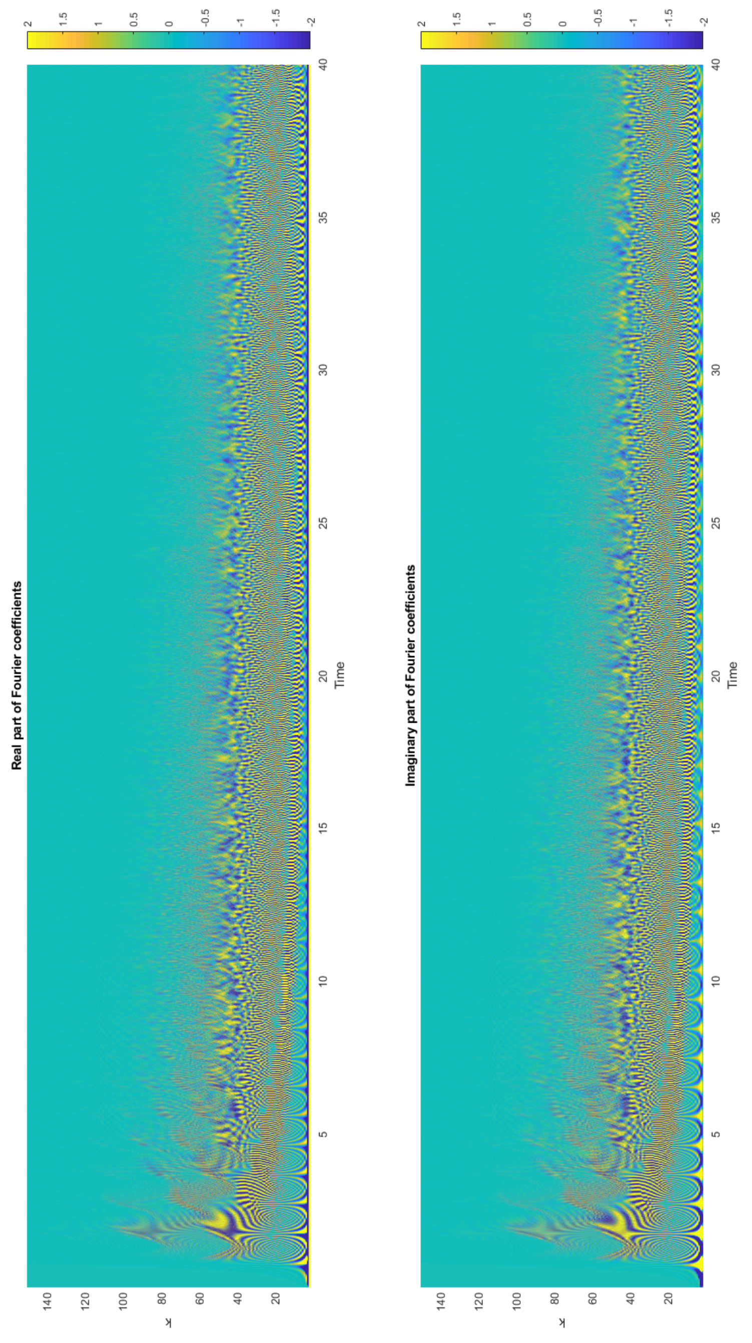

Figure A.3: Contourplot of the real part (left) and the imaginary part (right) of the Fourier coefficients on the time interval $[0, 80]$

Chapter B

Covariance and correlation matrices

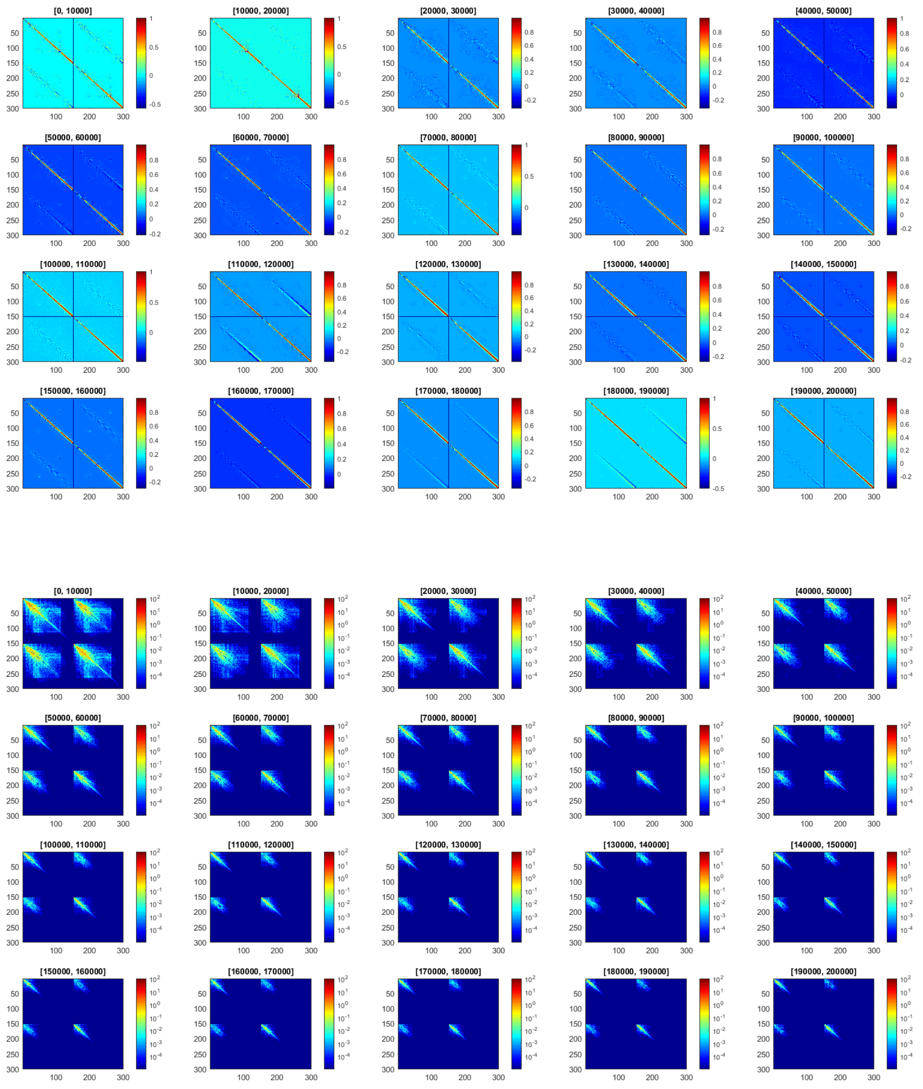

Figure B.1: The upper 20 plots show the correlation between the Fourier components of the solution of the forced burgers equation and the lower 20 plots show the covariance between those components. The coefficients have been split into a real and imaginary part. 1 through 150 indicates the real parts and 151 through 300 indicates the imaginary parts. Vertical and horizontal lines are NaN values.

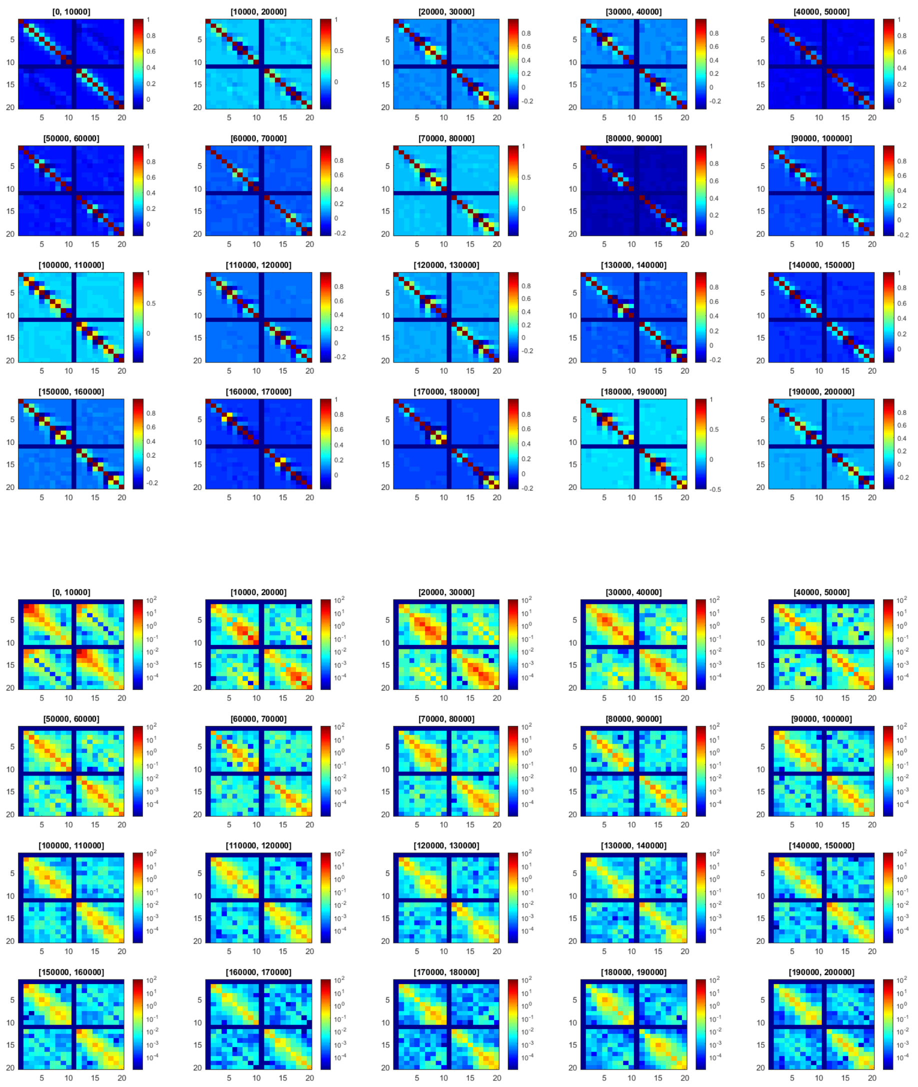

Figure B.2: This figure shows the same statistic as in figure B.1 expect that only the real and imaginary part of the first 10 wave numbers is shown. 1 through 10 indicates the real parts and 11 through 20 indicates the imaginary parts.

Bibliography

[1] Stephen Childress. *An introduction to theoretical fluid mechanics*, volume 19. American Mathematical Soc., 2009.

[2] Greg Norgard and Kamran Mohseni. A regularization of the burgers equation using a filtered convective velocity. *Journal of Physics A: Mathematical and Theoretical*, 41(34):344016, 2008.

[3] Wubs.F.W. Lecture notes: Computational methods of science, 2019/2020.

[4] TE Mulder. *A wavelet-based grid-switching method for energy cascades in Burgers' equation*. Master's thesis, Faculty of Science and Engineering, 2014.

[5] Jaideep Pathak, Zhixin Lu, Brian R Hunt, Michelle Girvan, and Edward Ott. Using machine learning to replicate chaotic attractors and calculate lyapunov exponents from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(12):121102, 2017.

[6] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks*, 20(3):335–352, 2007.

[7] Zbigniew Nitecki. The subsum set of a null sequence. *arXiv preprint arXiv:1106.3779*, 122, 2011.

[8] Mantas Lukoševičius. A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade*, pages 659–686. Springer, 2012.

[9] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.

[10] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

[11] Adrian Millea. *Explorations in echo state networks*. PhD thesis, Faculty of Science and Engineering, 2014.

[12] Peter J Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.

[13] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016.

[14] Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Physical Review Letters*, 120, 01 2018.

[15] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.

[16] Jaideep Pathak, Alexander Wikner, Rebeckah Fussell, Sarthak Chandra, Brian R Hunt, Michelle Girvan, and Edward Ott. Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 28(4):041101, 2018.

[17] Wybe Rozema. *Low-dissipation methods and models for the simulation of turbulent subsonic flow*. PhD thesis, PhD thesis, University of Groningen, 2015.

[18] Afroza Shirin, Isaac S Klickstein, and Francesco Sorrentino. Stability analysis of reservoir computers dynamics via lyapunov functions. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(10):103147, 2019.