Using a Physics-Informed Neural Network to solve the Ideal Magnetohydrodynamic Equations

by

J.F. Bouma

To obtain the degree of Bachelor of Science in Applied Mathematics and Applied Physics
at the Delft University of Technology

Student number:	4675053
Supervisors:	Dr. M. Möller (EEMCS)
	Dr. ir. D. Toshniwal (EEMCS)
	Dr. A.R. Akhmerov (TNW)
Other committee members:	Dr. J.L.A. Dubbeldam (EEMCS)
	Assoc. Prof. dr. S. Kenjereš (TNW)

Delft, July 2020

Abstract

In this work we investigate neural networks and subsequently physics-informed neural networks. Physics-informed neural networks are a way to solve physical models that are based on differential equations by using a neural network. The wave equation, Burgers' equation, Euler's equation, and the ideal magnetohydrodynamic equations are introduced and solved with physics-informed neural networks. The solutions to the first equations were captured well. The solution to the ideal magnetohydrodynamic equations contained some problems. These problems include transitions between different types of behaviour and exact values of constant sections. On the other hand, general shape and behaviour of the curve and locations of contact discontinuities were predicted well.

Contents

1 Introduction

Differential equations are the backbone of many mathematical and physical models. It is no wonder that mathematicians are always looking for new and better ways to solve differential equations. The most classical way to get a solution for a differential equation, is by solving it exactly. This means finding an equation that satisfies the differential equations and the boundary and initial conditions. This can be quite a hard task.

Currently, especially in physics, the most used method for obtaining the solution for a differential equation is by using a so-called numerical method. To be more precise, a method that discretises the domain of the function and subsequently discretises the differential equation. Then, linear algebra can be used to solve the discretised equation. An example of a method to do this, is the Finite Element Method.

An example of one of these models are the ideal magnetohydrodynamic equations. These equations model the behaviour of electrically conducting fluids, e.g. plasmas or liquid metals. A topic that is currently getting a lot of attention in both mathematics and computer science is artificial neural networks.

In 2019, Raissi et al. combined these topics, creating the field of physics-informed neural networks [12]. The goal of this thesis is to investigate whether an artificial neural network can solve the ideal magnetohydrodynamic equations.

In chapter 2, the basics of neural networks will be explained. The step to physics-informed neural networks will be made in chapter 3. The ideal magnetohydrodynamic equations will be discussed in chapter 4. In chapter 5, the results will be discussed. Finally, in chapter 6, a conclusion will be given. This thesis has been written as part of the double bachelor's degree Applied Mathematics and Applied Physics at the Delft University of Technology.

All the code that was used in this thesis can be found on `https://github.com/jort99b/beppinns`.

2 Artificial Neural Networks

2.1. Structure

The human brain is built up with neurons. These neurons are connected to other neurons with their dendrites and axons [4]. A human neuron is depicted in figure 2.1.

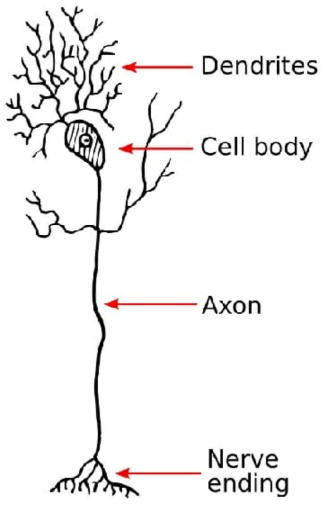

Figure 2.1: A human neuron. [4]

This is where the similarities between biological neural networks and artificial neural networks end. An artificial neural network (from here on out, simply neural network) is built up of layers. There are three types of layers, namely, the input layer, hidden layer, and output layer. There can be multiple hidden layers, depending on the complexity of the neural network. Each layer contains a certain number of neurons. In a neural network, a neuron is a node that assumes a certain value. Each neuron is connected to all the neurons in the previous layer. A schematic overview has been included in figure 2.2.

Each connection between neurons has a certain weight. This weight tells the neuron how much it should take over from the previous neurons. This is formalised as follows.

$$u_i = w_{i1}x_1 + ... + w_{im}x_m. \tag{2.1}$$

Where u_i is the i-th neuron in the layer we want to calculate, x_k is the k-th neuron in the previous layer, and w_{ik} is the weight corresponding to the connection between x_k and u_i. It can be seen that the previous layer has m neurons. Now we introduce a bias. This bias gives the neural network more flexibility. This will look as follows.

$$u_i = w_{i1}x_1 + ... + w_{im}x_m + w_{im+1}. \tag{2.2}$$

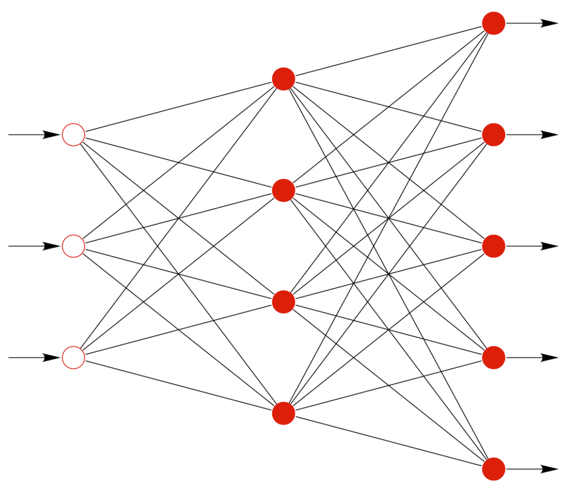

Figure 2.2: An artificial neural network. [14]

Where w_{im+1} is the bias. Equation 2.2 holds for every neuron in the layer we are interested in. Those fluent in linear algebra might have already seen the inner product or even the matrix calculations that will make for much nicer notation, as seen in the following equation.

$$\boldsymbol{u} = W\boldsymbol{x} + \boldsymbol{w}. \tag{2.3}$$

Where $\boldsymbol{u}$ is the vector of all neurons in the layer we are interested in, W is the matrix of all w_{ik} (upto index m), $\boldsymbol{x}$ is the vector of all neurons in the previous layer, and $\boldsymbol{w}$ is the vector with all biases. Note that W has m columns and that the number of rows is equal to the number of neurons in the layer that we are interested in.

Depending on the weights and biases, the values in the neurons can be anything. In order to normalise the behaviour of the neurons, a transfer function can be used. This can be almost any function. For example, if neurons may only take binary values, a Heaviside-function can be applied to every u_i to achieve good values. In most classical examples of neural networks, the sigmoidal function is used, which is defined as follows. [15]

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{2.4}$$

However, in most use cases, the hyperbolic tangent is a more practical approach. Currently, the most popular function is the rectified linear unit (ReLU), which is equal to $\max(0, x)$. Once chosen, the transfer function is fixed[11].

Now that we have determined the structure of the neural network and how it works, we want to start using it. This is as easy as performing all the calculations layer by layer. However, we quickly run into a problem, since the weights and biases are unknown.

2.2. Learning

Suppose we assign random values to all the weights. Then with a certain input, the output will most likely be nothing like the output we desire. This is why we first need to train the neural network. For this training, training data is required. This training data is comprised of inputs, that we already know the answer (output) for. Suppose we know that for a certain input $\boldsymbol{x}$, we know the correct output is $\boldsymbol{u}$. As established before, with randomly assigned weights, the output of the neural network will not resemble $\boldsymbol{u}$. We introduce the cost function:

$$(\boldsymbol{u} - \boldsymbol{v}) \cdot (\boldsymbol{u} - \boldsymbol{v}). \tag{2.5}$$

Where $\boldsymbol{v}$ is the output of the neural network. It is clear that this is simply the squared norm of the difference between the desired output and the actual output. It is the error we make in our neural network. The next step is now clear, we want to minimise this error.

In Calculus, when confronted with the task to determine a minimum of a function, the task is clear. Simply calculate the gradient and determine where it is equal to zero. However, depending on the size of the neural network, this can be quite a daunting task. We need to find a quick way to differentiate with respect to a lot of variables.

Automatic differentiation is a chain rule based technique to determine partial derivatives quickly [5]. This is crucial, because this needs to be done a lot. By applying the chain rule as often as possible, a derivative of a difficult function reduces to a product of the derivatives of elementary functions [9]. The derivative of the following function will be determined using automatic differentiation as an example:

$$z = \sin(8x + 3y) \tag{2.6}$$

Suppose we want to know $\frac{\partial z}{\partial x}$ at $(x, y) = (1,2)$. With automatic differentiation we first split this function into two parts: $u = 8x + 3y$ and $z = \sin(u)$. Then, in the so-called forward pass the following is done:

$$x = 1, y = 2,$$

$$u = 8x + 3y = 14,$$

$$z = \sin(u) \approx 0.9906.$$

Then in the backward pass the following is done:

$$\frac{\partial z}{\partial z} = 1,$$

$$\frac{\partial z}{\partial u} = \cos(u) \approx 0.1367,$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial u}\frac{\partial u}{\partial x} = \frac{\partial z}{\partial u} \cdot 8 \approx 1.0939.$$

Now that we have a way to calculate the gradient, we can minimise the cost function. This is done with an optimiser.

There are different types of optimisers. Each type of optimiser has its own merits. The two optimisers used in this paper are the Adam optimiser and the L-BFGS-B optimiser. The Adam optimiser is computationally efficient and can handle large amounts of data and parameters [8]. The L-BFGS-B optimiser estimates the inverse of the Hessian matrix. This way it is not necessary to compute the entire Hessian each iteration.

After we have trained our neural network, it is important to verify that we have found a good optimisation. This is done with a test data set. This test data set is similar to the training data set, but it is important to keep the two separate. If the learning can take the test data into account, the test is flawed. This test data is fed into the neural network. Then, the values for the cost function are determined. This way we can check whether our optimisation was good.

We now know the basics of how an artificial neural network works. The next step is to make the neural network a physics-informed neural network: get the physical model in the neural network.

3 Physics-Informed Neural Networks

3.1. Setup

There are several options to choose from when one wants to approximate a solution to a set of partial differential equations using a computer. Examples of this are the Finite Element Method, Finite Difference Method, or Finite Volume Method. All of these methods work by creating a grid on which we want to know the solution. Then using this grid the derivatives can be determined. For example, the first order derivative of f at a for a one-dimensional problem using the finite difference method is:

$$f'(a) = \frac{f(a+h) - f(a)}{h} \tag{3.1}$$

Where h is the grid size. All the needed derivatives for all the grid points are put into a matrix in this way. Then with the aid of linear algebra the values for $f(a)$ can be determined for all a on the grid. Then, if needed, this solution can be (polynomially) interpolated to achieve an approximate solution on the whole domain.

An attentive reader has noticed that the values for $f(a)$ are not known before trying to solve the matrix equation. This sounds a lot like the problem we discussed in chapter 2, where we wanted equation 2.5 to equal zero (or at least very small). Perhaps we can use a neural network to approximate the solution, since these are known to be good universal function approximators [7].

This is exactly what [12] has done. The neural network needs to know what equation it should solve. This can be done via the cost function. So we need to find a way to implement the given partial differential equations and its boundary conditions into a cost function.

[12] looked at the following type of nonlinear partial differential equations:

$$u_t + \mathcal{N}[u] = 0, x \in \Omega, t \in [0, T]. \tag{3.2}$$

Where $u(t,x)$ is the wanted solution, $\mathcal{N}$ is a nonlinear operator, and $\Omega \subseteq \mathbb{R}^D$ the domain where x lives. If we take $f(t,x) := u_t + \mathcal{N}[u]$ as shorthand, this equation can be made into a cost function quite easily with the mean squared error (MSE):

$$MSE = MSE_u + MSE_f. \tag{3.3}$$

Where

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u\left(t_u^i, x_u^i\right) - u^i \right|^2, \tag{3.4}$$

and

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f\left(t_f^i, x_f^i\right) \right|^2. \tag{3.5}$$

Where (t_u^i, x_u^i, u^i) are the given initial and boundary conditions. Equation 3.4 is taken over all the boundary and initial points and equation 3.5 is taken over all the points in the domain. So MSE_u ensures the boundary and initial conditions are met and MSE_f ensures the given differential equations are satisfied. It should be noted here that while MSE_u can be calculated fairly easily, determining $f\left(t_f^i, x_f^i\right)$ and thus MSE_f is more difficult. However, as discussed in the previous chapter, automatic differentiation is one of the qualities of a neural network. So we can use this same technique to compute the derivatives, without the need of making grids as in classical numerical methods. So f can also be determined relatively easily (albeit with a longer computing time then MSE_u).

Now we have the theoretical outline of how $u(x,t)$ can be approximated by a neural network. The procedure will be discussed in greater detail in section 3.4. The procedure of solving these equations with a neural network is called Physics-Informed Neural Networks (PINNs). Where physics-informed stands for the fact that the cost function consists of constraints given by physics. For example, symmetries, invariances, or conservation principles. In the most simple case as described above, these constraints are simply the differential equation and the initial and boundary conditions.

3.2. Burgers' equation

Multiple equations have already been solved with PINNs. [12] has preloaded the DeepXDE package with several examples, including Burgers' equation and Poisson's equation. An example of Burgers' equation has been included in figure 3.1. Here the following constraints were used:

$$f := u_t + uu_x - \frac{0.01u_{xx}}{\pi}, x \in [-1,1], t \in [0,1], \tag{3.6}$$
$$u(0,x) = -\sin(\pi x),$$
$$u(t,-1) - u(t,1) = 0.$$

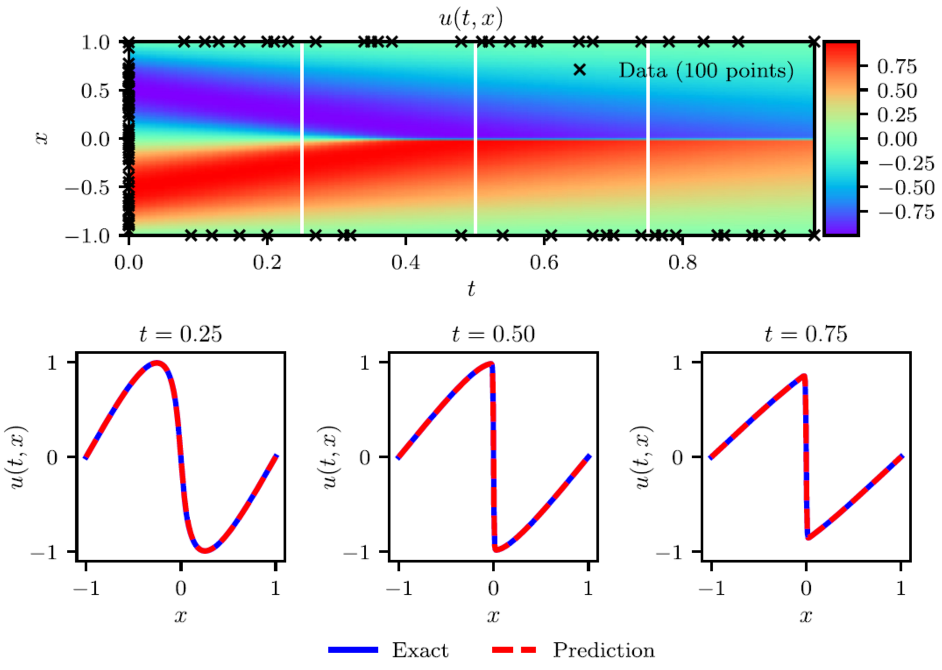

Figure 3.1: Burgers' equation: Top: Predicted solution, the points where the initial and boundary conditions were taken are indicated by crosses. Bottom: Several time-snapshots comparing the prediction to the exact solution. [cite Raissi et al]

3.3. Euler's equations

Another example that has been explored in a later paper by [10] set out to solve a more difficult set of equations, namely Euler's equations. The problem was defined as follows:

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho v\\ \rho E\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho v\\ \rho v^2+p\\ u(\rho E+p)\end{bmatrix}=0, x\in[0,1], t\in(0,2], \tag{3.7}$$

$$(\rho, v, p)_{t=0}=\begin{cases}(1.4,0.1,1.0), & \text{if } 0\le x<0.5\\ (1.0,0.1,1.0), & \text{if } 0.5<x\le 1\end{cases}.$$

With Dirichlet boundary conditions according to the given initial conditions. The exact solution to this problem is given by:

$$(\rho, v, p)(x,t)=\begin{cases}(1.4,0.1,1.0), & \text{if } x<0.5+0.1t\\ (1.0,0.1,1.0), & \text{if } x>0.5+0.1t\end{cases}. \tag{3.8}$$

The problem here is the travelling contact discontinuity. Two different approaches were taken. One with randomly distributed residual points and one with the residual points clustered around the travelling discontinuity. A problem that arises here is that the (approximate) location of the discontinuity does not have to be known. The results that were found have been included in figure 3.2.

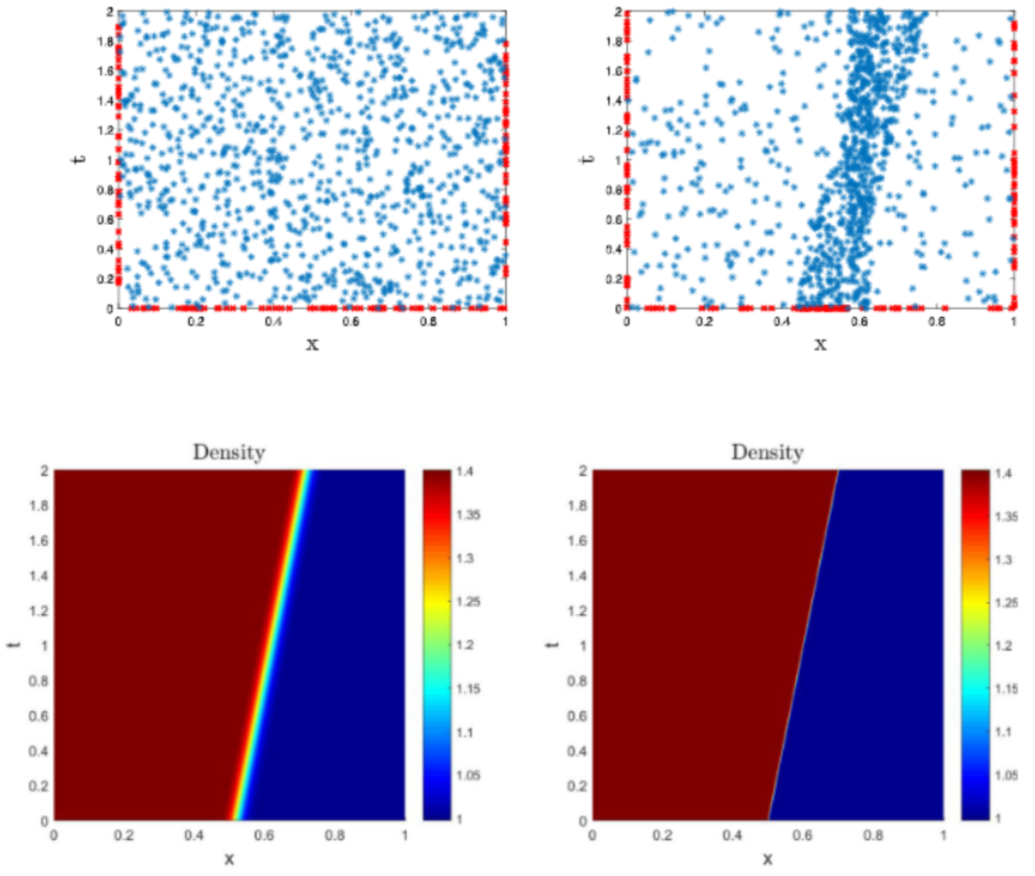

Figure 3.2: Euler's equations: Top: Distribution of the training and residual points, randomly distributed on the left, clustered around the travelling discontinuity on the right. Bottom: The found solutions for ρ, again randomly distributed on the left, clustered around the travelling discontinuity on the right. [10]

It can clearly be seen that when clustering the points around the discontinuity, the solution becomes much better. An accurate solution with randomly distributed residual points would require a much greater amount of residual points, resulting in longer computing times.

We aim to investigate whether a neural network can also solve more elaborate equations, such as the ideal magnetohydrodynamic equations. As will be explained in the next chapter, these equations are a combination of Euler's equations and Maxwell's equations. In these equations, multiple discontinuities can occur, causing more difficulties for the network.

3.4. DeepXDE

The PINNs in this project are based on DeepXDE [9]. DeepXDE is based on TensorFlow. TensorFlow is a machine learning Python package for creating neural networks [13].

When using DeepXDE, the first step is to define the differential equations that have to be satisfied. The equations can be simply typed out by using the gradients function from TensorFlow. After this, the boundary conditions and initial conditions can be defined. Then, the geometry has to be defined. There is a wide selection of geometries to choose from and a few more geometries have been made for use in this thesis. This can be easily done by modifying the source code.

After this all the data is gathered. This data, then gets combined with the type of neural network that is desired. Then, an optimiser can be chosen and the neural network is optimised. New methods for plotting the achieved solution have been implemented, namely, a three-dimensional plot and a slice plot.

Some limitations in the DeepXDE have been found. These limitations do not cause critical problems, but they make the process of implementing more time-consuming. The first limitation is that cross products cannot be used when defining the differential equations, because of this every vector product has to be written out by hand. The second limitation is that boundary and initial conditions can only be defined component-wise, not by defining a vector. Both these reasons cause the process of implementing high-dimensional vector-valued functions to be cumbersome.

4 Ideal Magnetohydrodynamic Equations

4.1. Equations

The ideal magnetohydrodynamic equations (ideal MHD equations) are a simple way to analyze fully ionized plasma dynamics with continuum mechanics. The plasma is assumed to be a homogeneous, quasi-electrically neutral, conducting fluid. It combines Maxwell's equations and Euler's equations for the electromagnetic part and fluid dynamics part respectively. [3]

The regular 3D ideal MHD equations in their conservative forms are as follows:

$$\begin{aligned}\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) &= 0, \\ \frac{\partial (\rho \boldsymbol{v})}{\partial t} + \nabla \cdot \left(\rho \boldsymbol{v} \otimes \boldsymbol{v} - \frac{1}{4\pi} \boldsymbol{B} \otimes \boldsymbol{B}\right) + \nabla p^* &= \boldsymbol{0}, \\ \frac{\partial E}{\partial t} + \nabla \cdot \left(\left(E + p^*\right) \boldsymbol{v} - \frac{1}{4\pi} \boldsymbol{B}(\boldsymbol{B} \cdot \boldsymbol{v})\right) &= 0, \\ \frac{\partial \boldsymbol{B}}{\partial t} - \nabla \times (\boldsymbol{v} \times \boldsymbol{B}) &= \boldsymbol{0}.\end{aligned} \tag{4.1}$$

Where ρ is the density of the fluid, $\boldsymbol{v}$ is the velocity of the fluid, $\boldsymbol{B}$ is the magnetic field, p^* is the total pressure, and E is the total energy. The total energy and the total pressure are defined as follows:

$$E = \rho \epsilon + \frac{1}{2} \rho \boldsymbol{v}^2 + \frac{1}{8\pi} \boldsymbol{B}^2, \tag{4.2}$$

$$p^* = p + \frac{1}{8\pi} \boldsymbol{B}^2. \tag{4.3}$$

Where ϵ is the specific internal energy and p is the pressure. The pressure is defined as follows:

$$p = (\gamma - 1) \left(E - \frac{1}{2} \rho \boldsymbol{v} \cdot \boldsymbol{v} - \frac{1}{2} \boldsymbol{B} \cdot \boldsymbol{B}\right) \tag{4.4}$$

Where γ is the ratio of specific heats, here taken to be 2.0.

It can be seen that they consist of eight differential equations with eight variables, determined by four coordinates. This makes them notoriously hard to solve, with the amount of calculation work quickly increasing.

It is interesting to note that, since there are no magnetic monopoles in our fluid, we know that $\nabla \cdot \boldsymbol{B} = 0$. This is not one of the equations that make up the ideal MHD equations, but rather something that follows from the other equations. This is called divergence-free involution.

Since the three-dimensional ideal MHD equations are incredibly hard to solve, this paper will focus on the one-dimensional ideal MHD equations. The 3D equations reduce to the 1D case as follows: [1]

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho v_x\\ \rho v_y\\ \rho v_z\\ B_y\\ B_z\\ E\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho v_x\\ \rho v_x^2+p^*-B_x^2\\ \rho v_x v_y-B_xB_y\\ \rho v_x v_z-B_xB_z\\ B_y v_x-B_x v_y\\ B_z v_x-B_x v_z\\ \left(E+p^*\right)v_x-B_x\left(B_x v_x+B_y v_y+B_z v_z\right)\end{bmatrix}=\mathbf{0} \tag{4.5}$$

The most important thing to note is that we went from eight differential equations, eight variables, and four coordinates to seven differential equations, seven variables, and two coordinates. This will make solving and comparing easier.

Again, since there are no magnetic monopoles, the divergence of the magnetic field is zero. However, since there is only one spatial dimension, this means that B_x is constant. Indeed, it can be seen that B_x is no longer a variable in the equations. This is also the reason that there are only seven variables left.

4.2. Brio-Wu Shock Tube

To test the performance of a solving scheme for the 1D ideal MHD equations, there exists a set of test problems. This test is called the Brio-Wu shock tube. The Brio-Wu shock tube is defined as the 1D ideal MHD equations with Dirichlet boundary conditions (so fixed constants at the edges) and the following initial conditions:

$$(\rho, v_x, v_y, v_z, B_y, B_z, p)_{t=0}=\begin{cases}(1.0,0.0,0.0,0.0,1.0,0.0,1.0), & \text{if } -1\leq x\leq 0\\ (0.125,0.0,0.0,0.0,-1.0,0.0,0.1), & \text{if } \quad 0< x\leq 1\end{cases} \tag{4.6}$$

Where the Dirichlet boundary conditions are determined by the initial conditions. The solution to this problem has been found by [2]. These results have been included in figure 4.1. The solutions have been shown for $t=0.2$. The intermittent solutions look like a shock wave, expanding from the discontinuity which contains all values that are obtained.

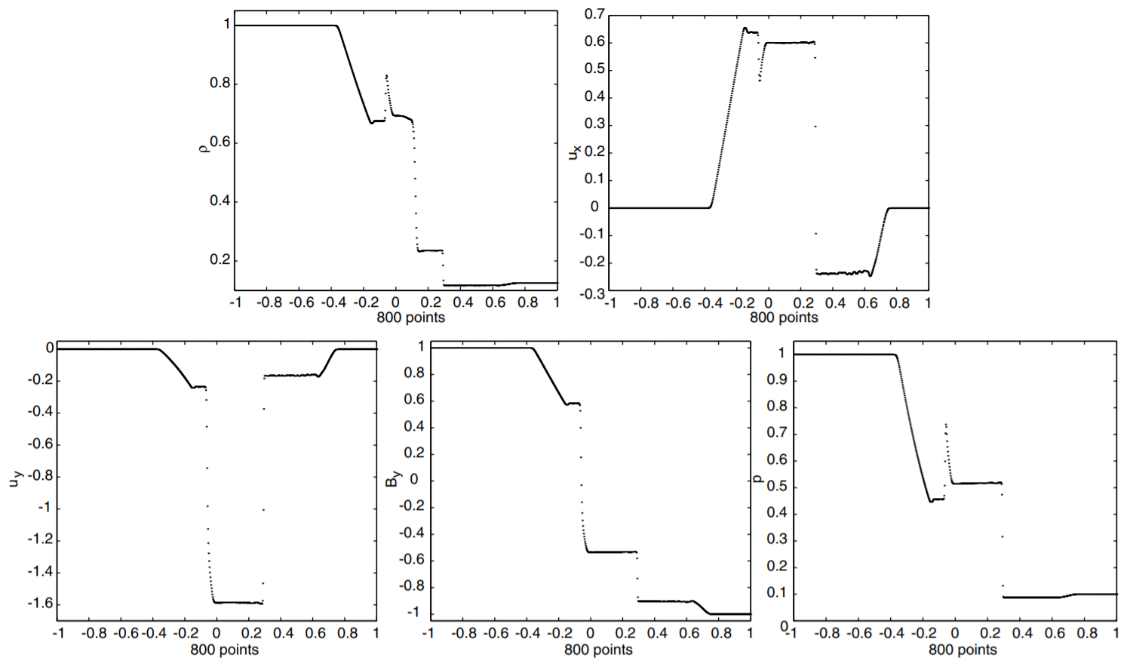

Figure 4.1: Solution to the Brio-Wu shock tube test problem for $t=0.2$, as found by [2]. From left to right, the graphs correspond to ρ, v_x, v_y, B_y, and p.

The graphs clearly show the some of the problems that can occur with the ideal MHD equations, namely the multiple discontinuities. These graphs have been obtained with a classic numerical solving scheme. It can be seen that with this method some minor mistake occur. For example, in the graph of v_x, for x between 0.3 and 0.5, some oscillatory behaviour occurs.

This problem is an excellent way to test the performance of a solving scheme, because the shocks are hard to solve correctly. This means that we can now use this problem to test the performance of our neural network.

5 Results

Unless otherwise indicated, all the results in this chapter were found by a neural network with 1000 training points for the differential equation and 600 for the boundary and initial conditions. Moreover, the neural networks have 3 hidden layers with 50 neurons each, the optimising procedure is performed with 20000 steps of the Adam optimiser and 5000 steps of the L-BFGS-B optimiser. The hyperbolic tangent is used as the transfer function.

5.1. Wave equation

The first result that is discussed here, is the one-dimensional wave equation. The one-dimensional wave equation is a great way to model the behaviour of a wave in a long rectangular box or a tensioned string. The equation goes as follows:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}.[6] \tag{5.1}$$

The program was tested with the following boundary and initial conditions:

$$u(-1,t) = u(1,t) = 0,$$
$$u(x,0) = -\sin(x),$$
$$\frac{\partial u}{\partial x}(x,0) = 0,$$

and c, the propagation speed, equal to 1. The solution to this problem can be determined quite quickly, with a variety of analytical methods. The solution is:

$$u(x,t) = -\sin(\pi x)\cos(\pi t) \tag{5.2}$$

This example of the wave equation is completely continuous and should thus, as seen in earlier works, pose no problem for the neural network. This can also be seen in figure 5.1.

This is clearly the correct solution. This can also be seen in the train loss and the test loss that the program gives. These are $2.51 \cdot 10^{-6}$ and $9.98 \cdot 10^{-7}$ respectively. To get an idea of how these values correspond to how accurately the solution depicts the analytical solution, the difference between the two has been depicted in figure 5.2.

So if the solution is continuous, the neural network can find a solution. Another interesting thing to look at is the development of the train loss and the test loss. These are set out in figure 5.3.

It can be seen that the first 20000 epochs with the Adam optimiser reduce the loss significantly at first, but slows down later. Then, at 20000, the loss spikes due to initialisation, but the following 5000 epochs with the L-BFGS-B optimiser reduce the error a bit more. It is peculiar that the train loss is larger than the test loss.

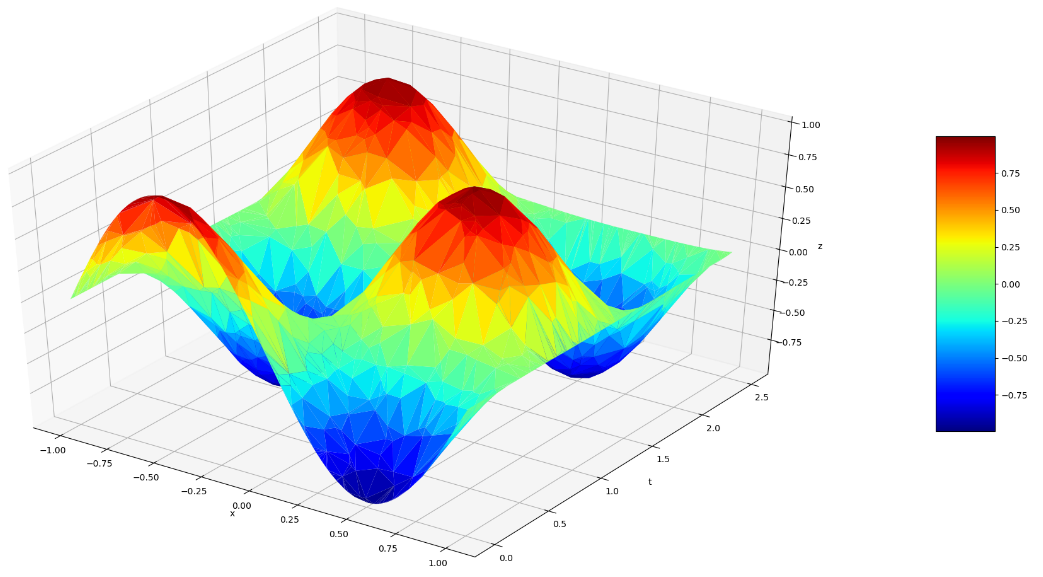

Figure 5.1: Solution to the wave equation as set out in equation 5.1.

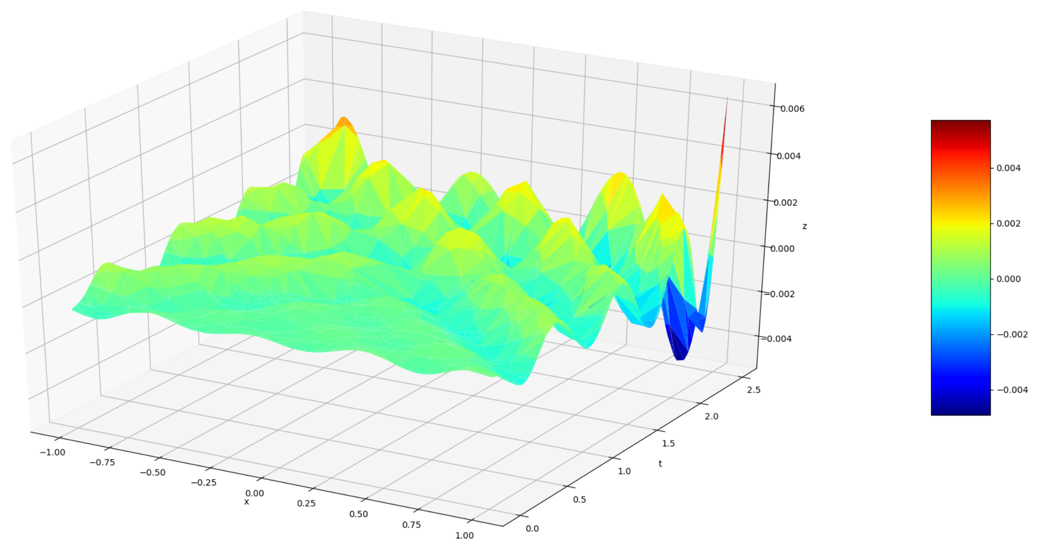

Figure 5.2: The difference between the analytical solution and the solution that was found with the neural network.

This is usually not the case for neural networks. The reason for this is unknown.

As said before, we use 1000 points in the domain and 600 points on the boundary. To get an idea of what this looks like, figure 5.4 has been included.

5.2. Burgers' equation

The next step up is to introduce a problem where a contact discontinuity occurs. The Burgers' equation is still a scalar-valued function on an (x,t)-domain. However, given continuous initial and boundary conditions, a contact discontinuity can develop. This will be more difficult for the neural network. We use the same example as in section 3.2. The results have been included in figure 5.5.

Comparing with figure 3.1, the solutions are equivalent. The train loss and test loss are $9.42 \cdot 10^{-6}$ and

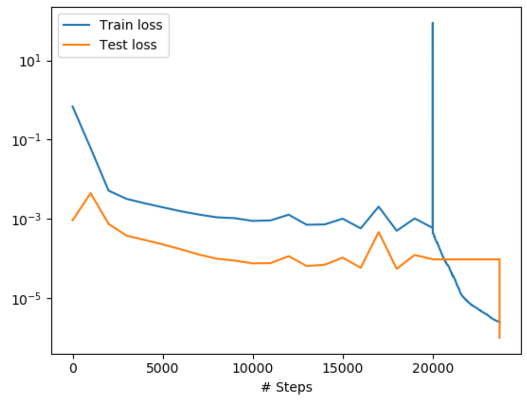

Figure 5.3: Development of the train loss and test loss for the solution of figure 5.1.

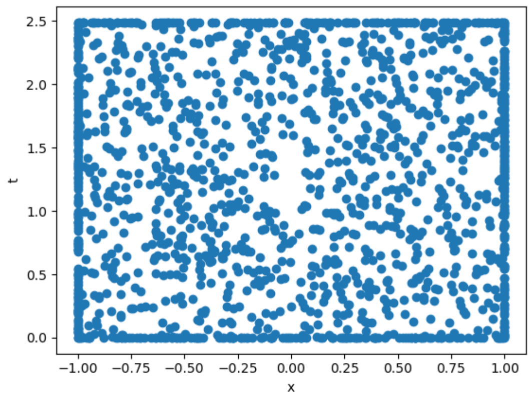

Figure 5.4: The domain and boundary points that were used for the solution of figure 5.1.

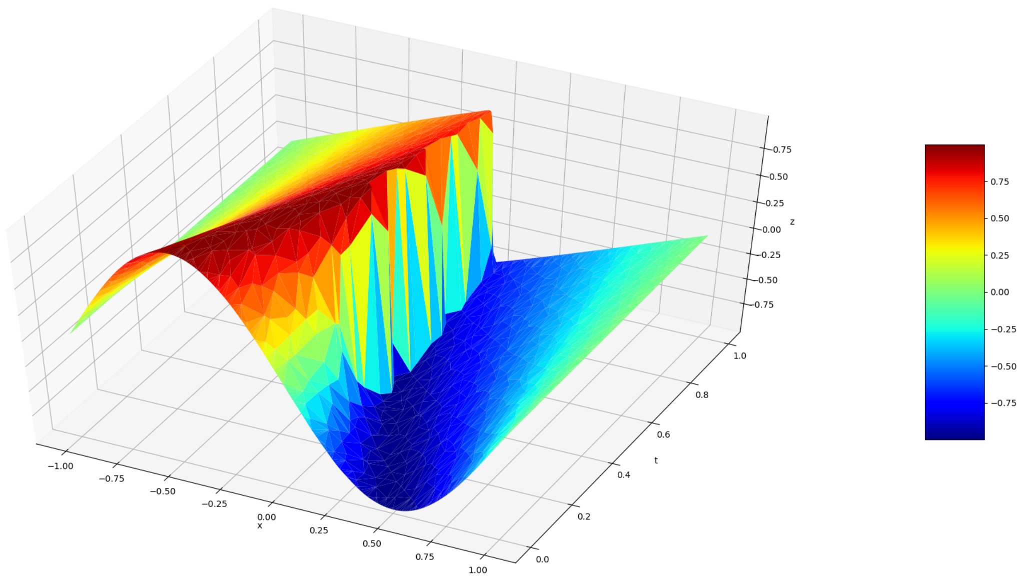

Figure 5.5: Solution to Burgers' equation as set out in section 3.2.

$7.83 \cdot 10^{-6}$ respectively. Indeed, we can see the gradual transition from sine to two discontinuous linear pieces. So now we know the neural network can handle contact discontinuities.

5.3. Euler's equations

Having introduced contact discontinuities, the next step is to include vector-valued functions. Euler's equations are also on an (x, t)-domain and can develop contact discontinuities. However, the function is vector-valued, it consists of density ρ, velocity v, and pressure p. This will again be more difficult for the neural network. We use the same example as in section 3.3, so we know the correct solution. First we implement it using the same technique as for the wave equation and Burgers' equation. Namely, randomly distributed points in the domain and randomly distributed boundary and initial points.

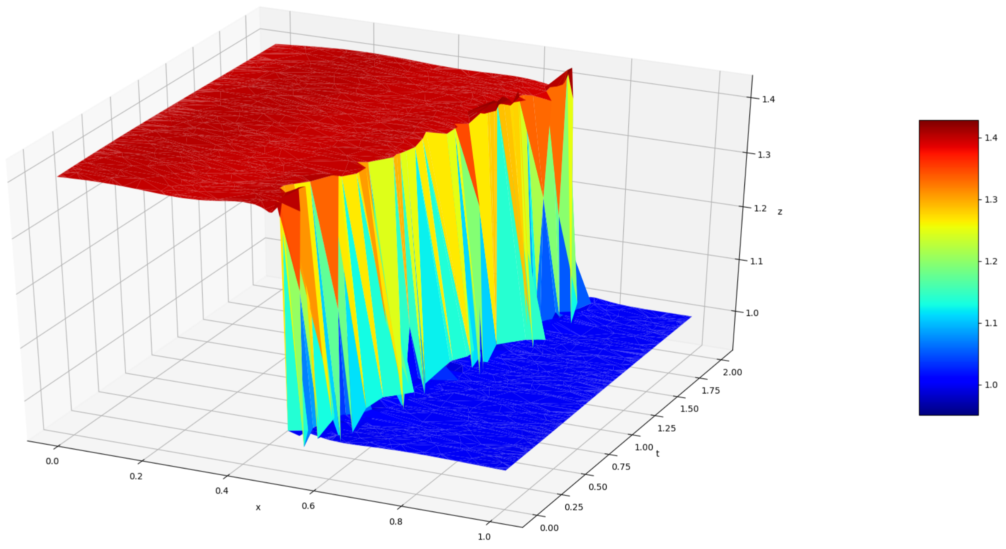

Figure 5.6: Solution for the density of Euler's equation as set out in section 3.3.

The correct solution can be recognised in the figure. However, not everything is entirely correct. The train loss and test loss are $1.33 \cdot 10^{-4}$ and $6.67 \cdot 10^{-5}$ respectively. This is quite a bit larger than the previous results. Some of the things that are worth noticing are the irregular edge of the contact discontinuity and the slight oscillating behaviour in the initial condition. This oscillating behaviour also travels through the solution.

A way to solve the oscillating behaviour in the initial condition might be to make the initial condition more important. This can be done by multiplying the error that is associated with the initial condition by a certain factor. The result of this is shown in figure 5.7. For now, a constant factor of 50 has been chosen.

The train loss and test loss are $2.91 \cdot 10^{-3}$ and $2.52 \cdot 10^{-3}$ respectively. The initial condition is satisfied very well now. However, because of the added weight to the initial condition, the neural network only focuses on the initial condition. This results in the differential equation not being satisfied very well. Different weights have been tried, but this either resulted in the oscillating behaviour or the differential equation not being satisfied well.

Since the oscillating behaviour only occurs near the contact discontinuity, the points that are close to the contact discontinuity are the ones that matter the most. So another way of solving the problem might be to specifically give those points a higher weight. This was attempted by weighting the error of the points with a normal distribution. This normal distribution is centered around the contact discontinuity. The result for this can be seen in figure 5.8.

The train loss and the test loss are $1.33 \cdot 10^{-3}$ and $1.11 \cdot 10^{-3}$ respectively. Again, the initial condition is satisfied well, but the differential equation is not. Trying different types of normal distributions was also to

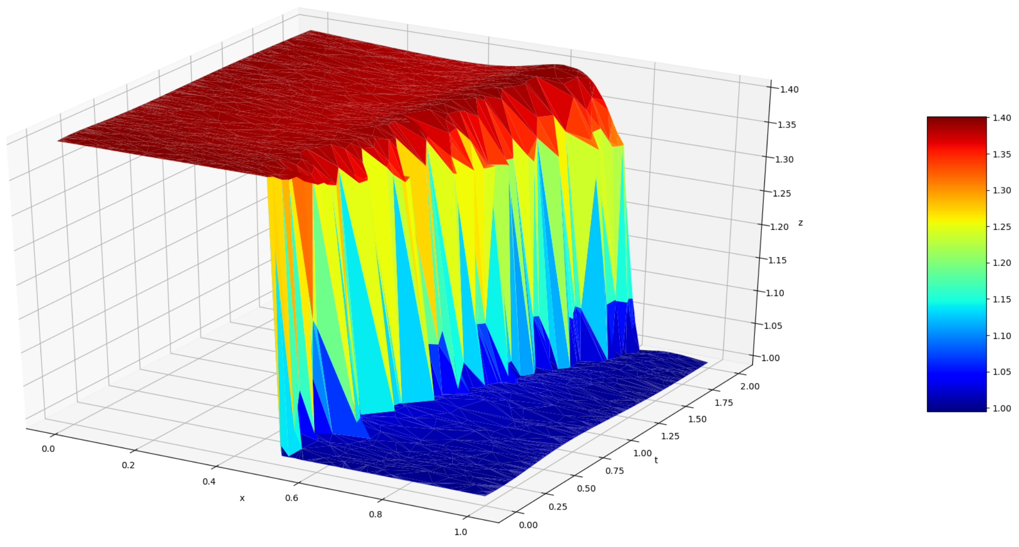

Figure 5.7: Solution for the density of Euler's equation as set out in section 3.3. However, the weight of the initial condition has been multiplied with 50.

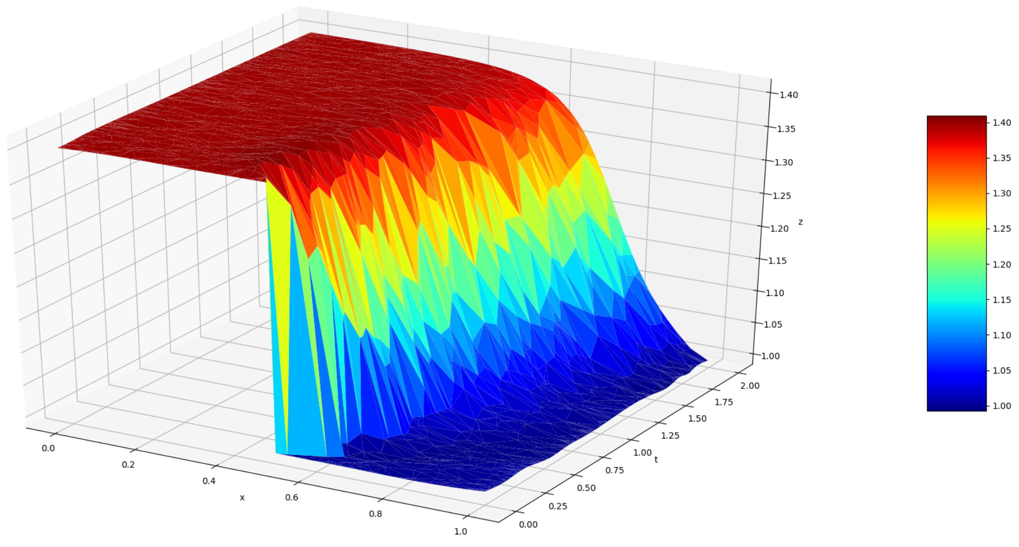

Figure 5.8: Solution for the density of Euler's equation as set out in section 3.3. However, the weight of the initial condition has been multiplied with a normal distribution, distributed according to the x coordinate.

no avail.

Another way to make the area around the contact discontinuity more important, is to simply assign more points to that area. This could be achieved by trying to find a solution with randomly distributed points. This will give us the rough position of the contact discontinuity. With this rough position, a new set of points can be created with more points around the discontinuity. This should also solve the problem of the irregular edge of the contact discontinuity. This has been tried in figure 5.10. Here, 100 points were sampled in the entire domain and 900 points were sampled in the neighbourhood of the contact discontinuity. A similar distribution was used for the boundary points. A figure of what this looks like has been included figure 5.9.

This results in a much better solution. This can also be seen in the train loss and test loss, which are $3.21 \cdot 10^{-5}$ and $1.65 \cdot 10^{-5}$ respectively. There is almost no oscillatory behaviour and the contact discontinuity

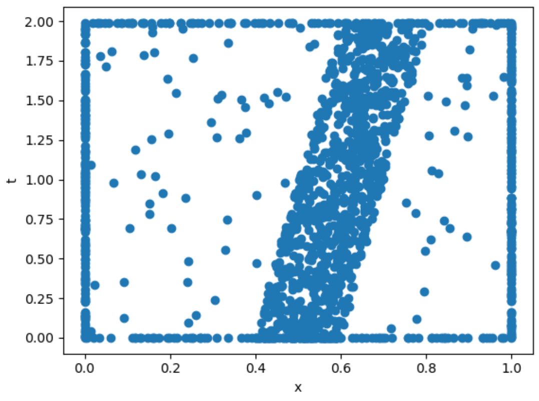

Figure 5.9: Distribution of points as used for the solution in figure 5.10.

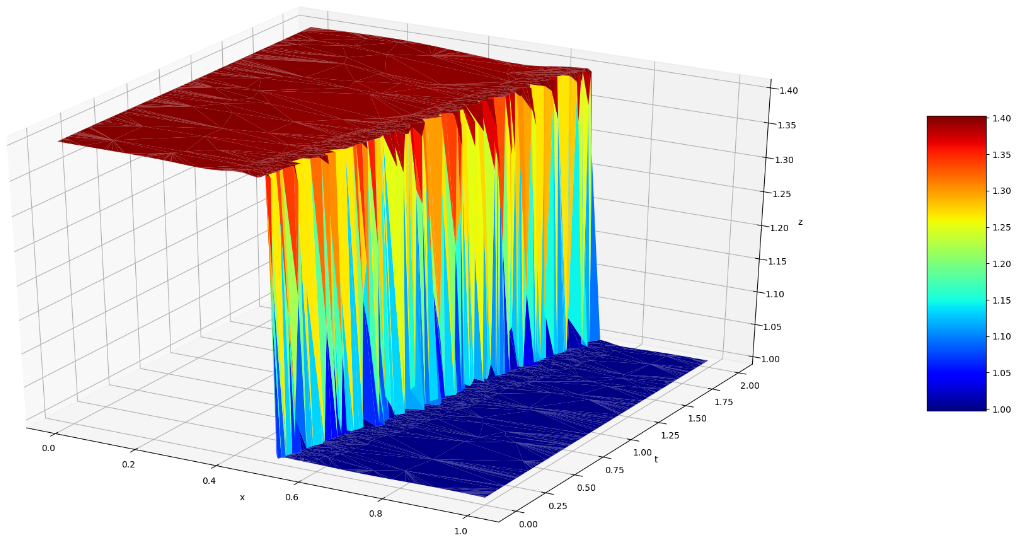

Figure 5.10: Solution of the density of Euler's equation as set out in section 3.3. However, rather than random distributed points, the points are clustered around the contact discontinuity. There are 100 points distributed over the whole domain and 900 in the cluster around the discontinuity.

has a sharp edge. It is important to note here that the total number of points was not changed, nor any of the other parameters, only the distribution of the points. A correct balance between points in the domain and points in the cluster area needs to be achieved here. Too few points in the cluster area will result in the problems described at the beginning of this section. However, too many points in the cluster area will result in a very sharp edge, but a bad solution in the rest of the domain, due to small errors.

5.4. Ideal MHD equations

Having computed vector-valued functions with contact discontinuities, the final step is to implement the ideal MHD equations. As discussed before, these are also vector-valued and develop contact discontinuities. However, rather than three variables, these have seven variables. This will result in longer computing times and potentially different results.

Again, the technique of the wave equation and Burgers' equation is used first. The result can be seen in figure 5.11.

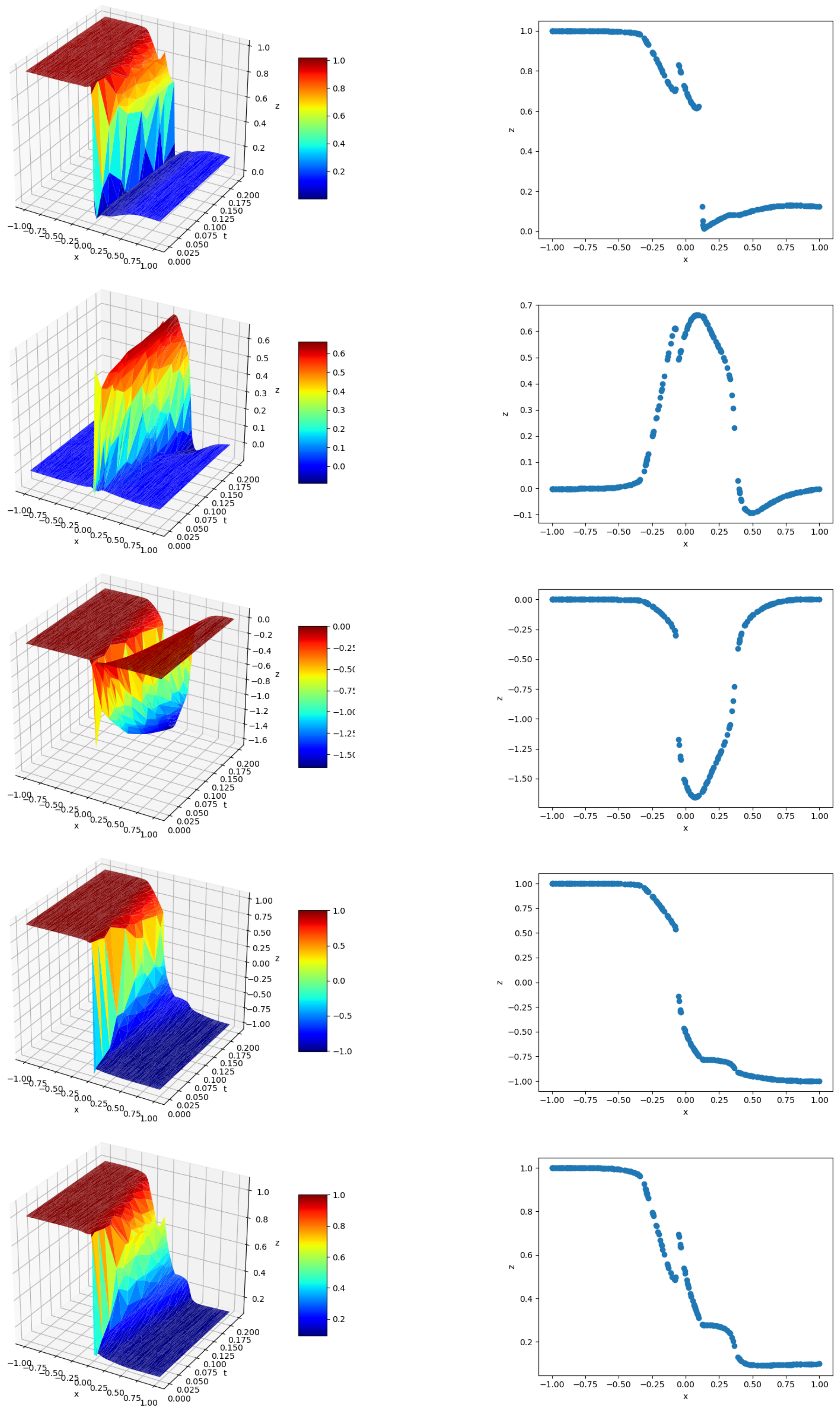

Figure 5.11: Solution of the Brio-Wu shock tube problem as set out in section 4.2. From top to bottom: ρ, v_x, v_y, B_y, and p (denoted as z on the axis). On the left the whole domain, on the right a snapshot at $t = 0.2$.

The train loss and test loss are $5.05 \cdot 10^{-3}$ and $2.00 \cdot 10^{-3}$. All the distinct parts of the Brio-Wu shock tube as seen in figure 4.1 can be recognised in the figure. However, two things stand out. The right side of the initial condition of the density ρ is not satisfied at all, slowly dipping too low as it goes in the direction of the contact discontinuity. Moreover, all the solutions are too smooth, not capturing the straight edges of the correct solution from figure 4.1.

Again, we first tried to mitigate these problems by weighting the initial condition. However, this did not produce any significant results. So we immediately move on to the tactic of clustering the points around the contact discontinuities. Again, a rough position of the cluster area was determined using the results where the points were randomly distributed. A graph of the points that were used is shown in figure 5.12. These points are not only clustered around the contact discontinuities, but also towards the initial condition. Meaning that the density of points at the initial condition is higher than at $t = 0.2$. This is to add more weight towards the initial condition. The results that were achieved with these points are shown in figure 5.13.

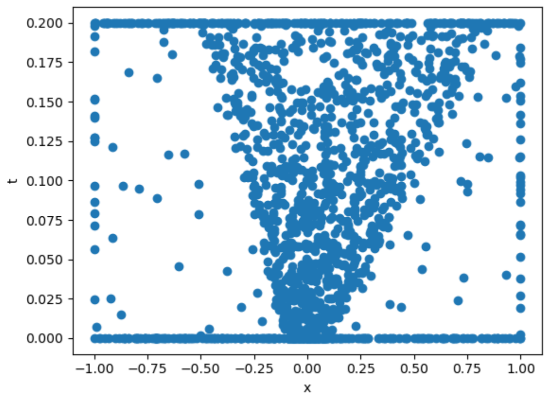

Figure 5.12: Distribution of points as used for the solution in figure 5.13.

The train loss and test loss are $3.76 \cdot 10^{-2}$ and $8.94 \cdot 10^{-3}$ respectively. When comparing these losses to the losses of the first solution, one could think that this solution is worse. However, this is not necessarily the case. Since there are more points around the contact discontinuities, the difficult area, one can expect that the average loss might be a bit higher.

When comparing the three-dimensional plots for v_x and v_y, the second solution is much steeper at $(0,0)$. This means that the solution goes to the correct height much quicker, which indicates a better solution. Also, it can be seen that the second solution goes up and down much faster than the first solution. In this case, this more closely resembles the correct solution.

However, when comparing the solutions for v_x in the snapshot, the behaviour at the right seems better in the first solution. Another important thing to note can be seen in the snapshot of p. Here the area right of the small peak in the middle should be higher than the area on the left. This is something neither of the solutions capture.

It is clear that some things are captured better than others. The general behaviour can be seen quite well. Discontinuities are also captured well. However, transitions from constant to linear are not captured accurately. A potential solution could be to include more points, however, this would drastically increase computing time. Another solution might be to add more physical constraints to the model, for instance preservation of mass or divergence-free involution for the three-dimensional case.

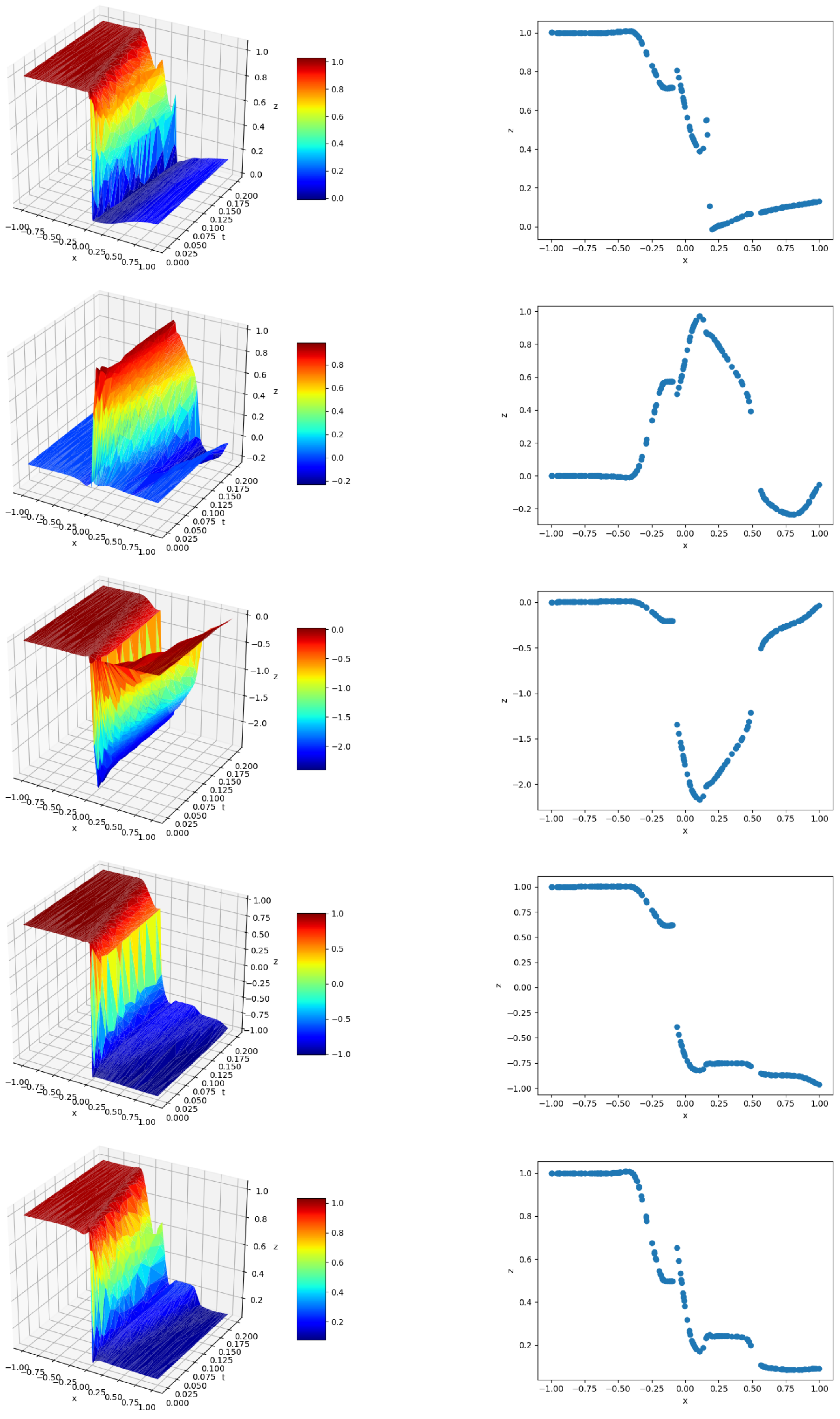

Figure 5.13: Solution of the Brio-Wu shock tube problem as set out in section 4.2. Here the domain and boundary points were clustered around the contact discontinuities. From top to bottom: ρ, v_x, v_y, B_y, and p (denoted as z on the axis). On the left the whole domain, on the right a snapshot at $t = 0.2$.

6 Conclusion

In this thesis, an overview of the way towards physics-informed neural networks was given. This was done by first introducing the general concept of artificial neural networks and then exploring how to incorporate differential equations into the neural network. After this the ideal magnetohydrodynamic equations were introduced, along with an important test problem called the Brio-Wu shock tube problem.

Subsequently, physics-informed neural networks were used to solve the wave equation, Burgers' equation, and Euler's equation. These equations slowly work their way to the ideal magnetohydrodynamic equations, in increasing difficulty.

Then, the Brio-Wu shock tube problem was attempted to be solved using two techniques. The results were mixed. The physics-informed neural network was able to predict the general shape and behaviour of the shock tube, as well as the locations of contact discontinuities, very well. However, some errors were made. A point of difficulty was transitions between constant and linear sections and not all values of constant sections were captured accurately.

Further research could be conducted into adding more physical constraints into the neural network. So, not only the differential equations, but for instance also preservation of mass or divergence-free involution.

Bibliography

[1] J. Balbás. One dimensional ideal mhd equations. http://www.csun.edu/~jb715473/examples/mhd1d.htm#density. Accessed: 2020-07-01.

[2] Tadmor E. Wu C.-C. Balbás, J. Non-oscillatory central schemes for one- and two-dimensional mhd equations: I. *Journal of Computational Physics*, 201:261–285, 2004.

[3] A Cimino, G Krause, S Elaskar, and A Costa. Characteristic boundary conditions for magnetohydrodynamics: The brio–wu shock tube. *Computers & Fluids*, 127:194–210, 2016.

[4] Biology Dictionary. Dendrite. https://biologydictionary.net/dendrite/, 2017. Accessed: 2020-06-16.

[5] A. Griewank. On automatic differentiation. 1988.

[6] R. Haberman. *Applied Partial Differential Equations with Fourier Series and Boundary Value Problems*. Pearson Education Limited, 2013.

[7] K. Hornik, M. Stinchcombe, H. White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[8] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[9] L. Lu, X. Meng, Z. Mao, and G.E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.

[10] Z. Mao, A.D. Jagtap, and G.E. Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[11] V. Nair and G.E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[12] Maziar Raissi, Paris Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[13] TensorFlow. Why tensorflow. https://www.tensorflow.org/about. Accessed: 2020-07-11.

[14] Universiteit van Amsterdam. Feedforward neural networks 1. what is a feedforward neural network? https://www.fon.hum.uva.nl/praat/manual/Feedforward_neural_networks_1__What_is_a_feedforward_ne.html. Accessed: 2020-06-19.

[15] J. Zupan. Introduction to artificial neural network (ann) methods: What they are and how to use them. *Acta Chimica Slovenica*, January 1994.