Approximation of Solutions of Governing Equations using Fourier Transform and Support Vector Machine

Ahmad Izzuddin[a], Muhammad Nursalman[a], Hadi Susanto[b], Lala Septem Riza[a]

[a]*Department of Computer Science Universitas Pendidikan Indonesia, Jl. Dr. Setiabudi No.229, 40154, Bandung, Indonesia*
[b]*Department of Mathematics Khalifa University, Shakhbout Bin Sultan St - Hadbat Al Za'Faranah, 127788, Abu Dhabi, United Arab Emirates*

Abstract

Numerical models of systems are a crucial part of science and engineering. The use of machine learning in this space for operator learning provides an alternative as data-driven surrogates. The Fourier Transform provides a key component for learning the relationship between a function and its derivatives. Building on Spectral Neural Operators (SNO), we propose a Support Vector Machine (SVM) based framework to learn the underlying governing equations of a system based on data. We study the viability and interpretability of the proposed framework on the derivative equation and the Burgers' equation. The model is able to learn from mathematically correct random data and is able to partially generalize to an exact solution of the Burgers' equation. The learned model is interpreted and verified to have learned the correct contributions of the input function coefficients to the output function coefficients.

Keywords: Operator Learning, LSSVR, PDE

2000 MSC: 35-04, 68T99

Email addresses: ahmadizzuddin@upi.edu (Ahmad Izzuddin), mnursalman@upi.edu (Muhammad Nursalman), hadi.susanto@ku.ac.ae (Hadi Susanto), lala.s.riza@upi.edu (Lala Septem Riza)

1. Introduction

Governing equations have been a critical part of the technological revolution. They help us understand better the systems that exist in our world (Braun, 1993). Typically, governing equations are differential equations. For systems with many variables, they are described by Partial Differential Equations (PDE). For example, the spread of heat on a cooking surface is described by the heat equation, with the temperature as a function of time and space. Differential equations are used because often, the behavior systems are easier to describe in the way they change.

Typically, when modeling a system with PDEs, there are different problem types based on the conditions that are imposed on the solution. The main problem types are where the solutions are constrained by its boundaries. For equations with a temporal component, an initial value can be imposed as a restriction on solution's value at that initial point in time. This problem is called the Initial Value Problem (IVP). The Boundary Value Problem (BVP) on the other hand, defines the restrictions on the solution at the spatial boundary of the system being simulated, such as values or derivatives the solution must have at the boundary. The Initial-Boundary Value Problem (IBVP) combines both conditions imposed on the solution. These problems are considered part of the larger group of forward problems. This is because, the objective is to determine the system's response in terms of the solution towards causal factors, namely parameters and imposed conditions (Groetsch, 1993; Vogel, 2002). The inverse problem, on the other hand, is the reverse process where parameters or conditions such as the initial value is the objective. Simply put, in inverse problems, the effects are used to compute the cause that led to it.

While some PDEs have analytical solutions, they are restrictive in ways such as the initial condition, or other parameters (Selvadurai, 2000; Kopriva, 2009; Olver, 2014; Schiesser, 2012; Wazwaz, 2010). Because of this, modeling a system with

differential equations usually involve the use of numerical methods. With their long history, there are now several well established general methods such as Finite Difference Method (FDM), Finite Element Method (FEM), and the Spectral Method. With mesh based methods such as FDM, evaluating the solutions at higher resolutions means either recomputing the solution with a higher resolution which increases the computational cost, or interpolation which can depend on the problem. For multiscale problems like Numerical Weather Prediction (NWP), the complex interactions are very difficult to model due to the vast differences in scale (Frank et al., 2024). Also, the user needs to know the equations involved before modeling the system. Which may not be available in some fields like ecology (Holmes et al., 1994; Turchin, 2001) and epidemiology (Brauer et al., 2019).

These challenges have spurred interest in modeling systems with the use of Machine Learning (ML) as surrogates. There are two types of modeling with ML models. The first kind is akin to function regression, where the model predicts solution values from coordinates and parameters. An example, Raissi et al. (2019) proposed Physics-Informed Neural Networks (PINN). The specific implementation they used is a Feedforward Neural Network (FNN) as an approximation for the solution. To enforce the PDE, the loss is computed with PDE residuals and other constraints. The partial derivatives for the residual are computed with ease using the automatic differentiation with respect to the input coordinates. Other works utilize convolutional neural networks (CNN) to compute the solution from input functions such as forcing terms or initial conditions. This approach generally means discretizing the functions on a grid and using these as training data. One study by Wang et al. (2020) predicts turbulent flow using spatial and temporal decomposition and a specialized U-Net, an architecture based on CNNs, to predict the velocity field from the decomposition of the previous velocity field. Part of the loss function is a regularization term for zero divergence in the velocity field to enforce incompressible fluid flow. This term was calculated using finite differences since auto differentiation cannot be used without the coordinates in the model

inputs. While the use of CNNs mean that discretization is implied, solutions of different initial conditions or parameter functions can be computed by inference and no retraining is required. This property is especially useful for many-query problems such as computing gradients for inverse problems.

This is where operator learning models that predicts basis function coefficients come in. Basis functions are collections of functions that share common properties and can be used to represent other functions in a linear combination. For example, the sine and cosines in the real Fourier series. This is the concept used by Fanaskov and Oseledets (2023) for their proposed model, termed Spectral Neural Operator (SNO). The model Neural Network (NN) is trained on features of input function coefficients which are computed using Fourier or Chebyshev transforms and labels of output function coefficients using the same transforms. This separates the concern of discretization from the problem of learning the relationships enforced by some system between functions. Du et al. (2024) extends the concept of mapping coefficients by proposing residuals in the spectral domain and leveraging Parseval's Identity to compute the spectral analog to the loss term in PINNs. This allows for self supervised learning in the spectral domain. The same benefits incorporating physics into PINNs also apply here without the pain points introduced by discretized model inputs and outputs. Another approach by Lu et al. (2021), essentially uses learned basis functions in the form of NNs. This has the benefit of the basis functions being custom-made to fit the problem and equations being modeled (Meuris et al., 2023). These methods show promise and provide solutions to parameters via the relatively fast process of inference.

The use of NNs does come with its own downsides. First, the loss functions used in their optimization represents a landscape with many local minima. This problem becomes worse with the addition of PDE residual terms that cause disparities in the gradients of each individual loss term and (Rathore et al., 2024; Krishnapriyan et al., 2021; Basir and Senocak, 2022). This has motivated the use of ML algorithms with a convex loss landscape for modeling. Support Vector

Machines (SVM) are one such family of algorithms (Vapnik, 2000). The appeal of SVMs is the fact that the model is formulated as a quadratic programming problem. This means there are strong guarantees for convergence, generalization, and complexity. Another formulation called Least-Squares Support Vector Machines (LSSVM) reformulates the problem as a linear system (Suykens, 2005). This leads to an easier problem that can be computed faster by well established algorithms like the many implementations of least squares solvers. Another advantage of the linear formulation is that this can be easily parallelized to exploit hardware like graphics processing units more widely known as GPUs in contrast to the commonly used Sequential Minimal Optimization (SMO) used for SVMs with quadratic objective functions.

An early work using SVMs to solve PDEs by Youxi Wu et al. (2005) introduced a method for solving the forward problem of Electro-Impedance Tomography. This work solved for the mathematical model of EIT which is given by Maxwell's equations by modeling the trial function as using an ϵ-SVR model. Another approach much more similar to PINNs was presented by Mehrkanoon and Suykens (2015). The residual and initial/boundary conditions are imposed as equality constraints on an LSSVM objective function. A different study by Leake et al. (2019), the incorporation of physics into the model is done slightly differently by utilizing the theory of functional connections to directly embed constraints into the solution. This means that the proposed method would satisfy the boundary condition exactly. These approaches, however, only learn a function regression problem constrained by the PDE. Meaning they are also not practical for many-query problems.

In this paper, we propose an operator learning model based on basis functions and LSSVM for regression, named SpectralSVR. Here we will focus on demonstrating the model's capability, starting with a proof of concept with the simple derivative equation. And then, the nonlinear PDE solving capabilities will be shown by approximating solutions to the Burgers' equation.

2. Methods

This section describes the proposed method and the case studies we used.

2.1. SpectralSVR

The design of the proposed model is built upon performing regression in the Fourier domain. This means that any function values that will be used by the model, should be transformed into their coefficients. And, any predictions from the model, which are coefficients, should be transformed back into function values.

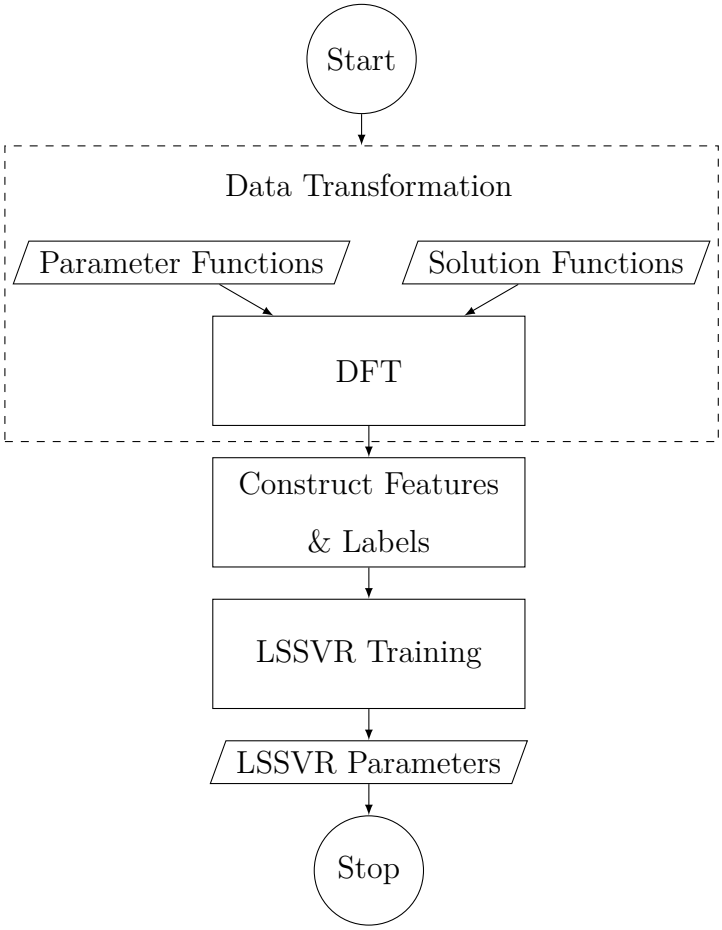


Figure 1: SpectralSVR training diagram.

The training process is shown in figure 1. The process in psuedocode is also shown in algorithm 1. The process starts with determining the parameter and

solution functions. Typically, the parameter functions would include the forcing term and others such as the initial solution function values. The solution function typically is the state variable that the PDE constraints using its derivatives, such as the displacement of a membrane for the wave equation. Next, the features and labels are constructed. Generally, the Fourier coefficients for both functions are computed using the DFT. And then, the parameter and solution coefficients are used as the feature and labels respectively. To finalize the construction, the coefficients are flattened into one dimension for each sample. And then, the features are scaled for training the LSSVR. Finally, after the feature and label construction, we train the LSSVR by computing the parameters of the model. In the psuedocode this starts by computing the kernel matrix from the features. And then the two matrices, $\mathbf{A}$ and $\mathbf{B}$, are put together to compute the model parameters via least-squares, which marks the end of the training process.

Algorithm 1 SpectralSVR Training Psuedocode.

procedure TrainSpectralSVR($\mathbf{U}, \mathbf{F}, C$)
  $\hat{\mathbf{U}} \leftarrow DFT(\mathbf{U})$
  $\hat{\mathbf{F}} \leftarrow DFT(\mathbf{F})$ ▷ Compute Fourier coefficients of training functions
  $\mathbf{X} \leftarrow finalizeFeatures(\hat{\mathbf{U}}, \hat{\mathbf{F}})$
  $\mathbf{y} \leftarrow finalizeLabels(\hat{\mathbf{U}}, \hat{\mathbf{F}})$
  $\Omega \leftarrow K(\mathbf{X})$ ▷ Construct matrix of inner products
  $\mathbf{A} \leftarrow [[]]$
  $\mathbf{B} \leftarrow [[]]$
  $\mathbf{A}[1:, 1:] \leftarrow \Omega + \frac{\mathbf{I}}{C}, \mathbf{A}[0, :] \leftarrow 0, \mathbf{A}[:, 0] \leftarrow 0$
  $\mathbf{B}[0] \leftarrow 0, \mathbf{B}[1:] \leftarrow \mathbf{y}$
  $\mathbf{S} \leftarrow leastSquares(\mathbf{A}, \mathbf{B})$
  $\alpha \leftarrow \mathbf{S}[1:], \mathbf{b} \leftarrow \mathbf{S}[0]$ ▷ Learned model parameters
  return $\alpha, \mathbf{b}$
end procedure

The prediction process is shown in figure 2. The process starts by transforming the feature functions into their coefficients with the DFT. These coefficients are then scaled with the scaling function fitted during training. The scaled feature coefficients are then used with the learned parameters to produce the predicted coefficients using the LSSVR prediction function. Finally, the inverse DFT is computed on the predicted coefficients to obtain the predicted function value.

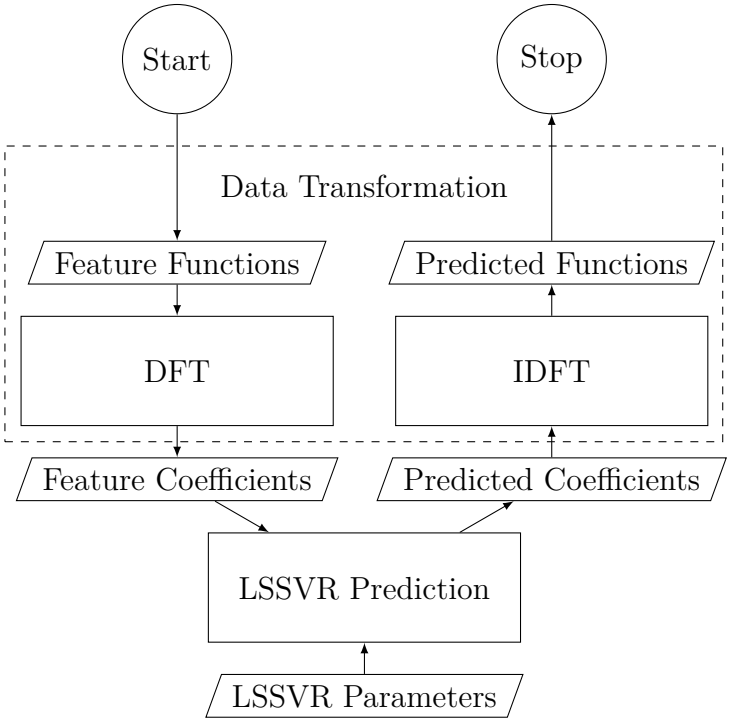


Figure 2: SpectralSVR prediction diagram.

2.2. Case Studies

The model's ability will be verified and demonstrated using two equations. The data will be generated using the method of manufactured solutions (Roache, 2002; Salari and Knupp, 2000; Vedovoto et al., 2011). First, the model will be tasked to learn the basic relationship defined by the simple derivative equation (1). The solution functions are generated as Fourier polynomials with random coefficients sampled from a normal distribution. The generated solution function takes

the form shown in equation (2). The coefficients of the derivative function in equation (3) is computed using equation (5).

$$\frac{\mathrm{d}u\left(t\right)}{\mathrm{d}t} = a\left(t\right) \tag{1}$$

$$u\left(t\right) = \sum_{k} \hat{u}_k e^{2\pi ikt/T} \tag{2}$$

$$a\left(t\right) = \sum_{k} \hat{a}_k e^{2\pi ikt/T} \tag{3}$$

$$\sum_{k} \hat{u}_k \times \left(2\pi ik/T\right) e^{2\pi ikt/T} = \sum_{k} \hat{a}_k e^{2\pi ikt/T} \tag{4}$$

$$\hat{a}_k = \hat{u}_k \left(2\pi ik/T\right) \tag{5}$$

In total, we generate 5000 unique functions with 50 complex coefficients each. We also add noise to the function values at three levels of the average function standard deviation. The entire set has three versions with different levels of noise which are 5%, 10% and 50%.

The second problem the model will be presented with is solving the Burgers' equation by stepping through time. Because of the nonlinearity of this equation, it is often used to test the capability of numerical solvers (Wood, 2006; Wazwaz, 2010; Kopriva, 2009). The formulation of the Burgers' equation we use is the forced viscous Burgers' equation in 1-dimension as shown in equation (6) where the viscosity is ν.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = f \tag{6}$$

Data generation is done similarly to the derivative equation. With the solution as in equation (7) and the forcing term equation (8), the equation for the forcing term is shown in equation (9). The nonlinear term is approximated, for efficiency,

in the physical domain and the coefficients subsequently computed as $\hat{uu}_k$.

$$u(x,t) = \sum_{k_x} \sum_{k_t} \hat{u}_k e^{2\pi i(k_x x/L + k_t t/T)} \tag{7}$$

$$f(x,t) = \sum_{k_x} \sum_{k_t} \hat{f}_k e^{2\pi i(k_x x/L + k_t t/T)} \tag{8}$$

$$\hat{f}_k = (2\pi i k_t/T)\hat{u}_k + (2\pi i k_x/L)\hat{uu}_k - \nu(2\pi i k_x/L)^2 \hat{u}_k \tag{9}$$

The functions generated for the Burgers' equation will use three different values of ν to in order to be able to analyze how the model behaves with varying levels of stiffness and nonlinearity. The values for viscosity are 0.0 for the inviscid equation, 0.01, and 0.1 for a more stiff PDE. For each viscosity level, we generate 500 unique solution functions and compute the corresponding forcing term using equation (9). Each function has 8 modes in space and 8 in time.

3. Results and Discussion

In this section, we will present the results of the experiments on the derivative equation and the Burgers' equation. The metrics we will show in this section are evaluated between the function values evaluated from the predicted coefficients and the function values of the test label coefficients.

3.1. Scenario 1: Derivative Equation

For the derivative equation, metrics between the noisy targets and predictions are shown in table 1. The metrics shows that the model is able to generalize from the training data and learn the simple linear antiderivative operator. Focusing on the R^2 scores, we can see that the model is off by 0.04 to the perfect score of 1.0 for the lowest noise level. While increasing noise levels does degrade the performance, this is partly due to the target values also having been perturbed.

For the function values of the 50% noise dataset, the values range on average from -2.5 to 2.49. This was approximated since there is no direct way to compute the amplitude of the function from just the coefficients. Here we used the inverse

transform and extracted the maximum and minimum values of each function from the discrete values. Since the RMSE in table 1 for the 50% noise dataset is 0.63, the error ratio comes to 0.126. One side note about table 1 is that the kernel scaling hyperparameter is the same for all noise levels because of the scaling which is 10.

Table 1: Performance metrics of function value from evaluated coefficient prediction in scenario 1 by noise level.

Noise level	MSE	RMSE	MAE	R^2	sMAPE
5%	0.03	0.18	0.15	0.97	0.39
10%	0.08	0.29	0.23	0.92	0.55
50%	0.62	0.79	0.63	0.49	1.05

The metrics between the predictions and the noise-free version of the target function values can be seen in table 2. The metrics show that the model predictions are slightly closer to the unperturbed functions compared to the perturbed versions. This can be seen as the influence of the independent measurement noise in the perturbed targets being removed from the testing metrics. The metrics show that the model performs slightly better with more pronounced differences for the higher noise levels.

Table 2: Performance metrics of function value from evaluated coefficient prediction compared to unperturbed targets in scenario 1 by noise level.

Noise level	MSE	RMSE	MAE	R^2	sMAPE
5%	0.03	0.18	0.14	0.97	0.38
10%	0.07	0.27	0.21	0.93	0.53
50%	0.37	0.61	0.49	0.62	0.94

The next part of this scenario is the results from predicting the exact solution of

$f(t) = 2\pi\cos(2\pi t)$ which is $u(t) = \sin(2\pi t)$. The function values will be computed, and the values put through the preprocessing steps as all other function values. The results are presented in tables 3. For 5% and 10% noise levels, the model shows that it has learned the relation. However, the 50% noise level, the model has been unable to predict the results well enough. The marked difference in error across all metrics between the 50% and the lower noise levels is more apparent for this specific exact problem compared to the test sets. A clear picture of this can be seen when we compare the sMAPE metric. For the test set, the predictions on average results in a value of 1, however, for this exact problem the sMAPE value is 0.23 to 1.65.

Table 3: Performance metrics of evaluated function values of coefficient prediction of exact antiderivative in scenario 1 by noise level.

Noise level	MSE	RMSE	MAE	sMAPE
5%	0.00	0.06	0.05	0.23
10%	0.03	0.18	0.15	0.36
50%	0.41	0.64	0.58	1.65

To test the generalization capabilities of the model, we use the sine function with a period of one shown in equation (11). The derivative is simply equation (10). The functions are discretized and noise is added. Finally, the values go through the same transformations to become feature and target coefficients.

$$f(t) = 2\pi\cos(2\pi t) \tag{10}$$

$$u(t) = \sin(2\pi t) \tag{11}$$

The predictions of the exact equation is shown in figure 3. Visually, we can see that as the noise level increases, the model predictions become worse. Another observation is how the higher the noise level, the more the predicted functions become closer to zero. This is explained by the double penalty phenomenon

(Lledó et al., 2023). The loss function used for LSSVR penalizes the model for predicting a non-zero value that turns out to be wrong compared to predicting a zero value. This results in a model with predictions that are close to the mean of the training data.

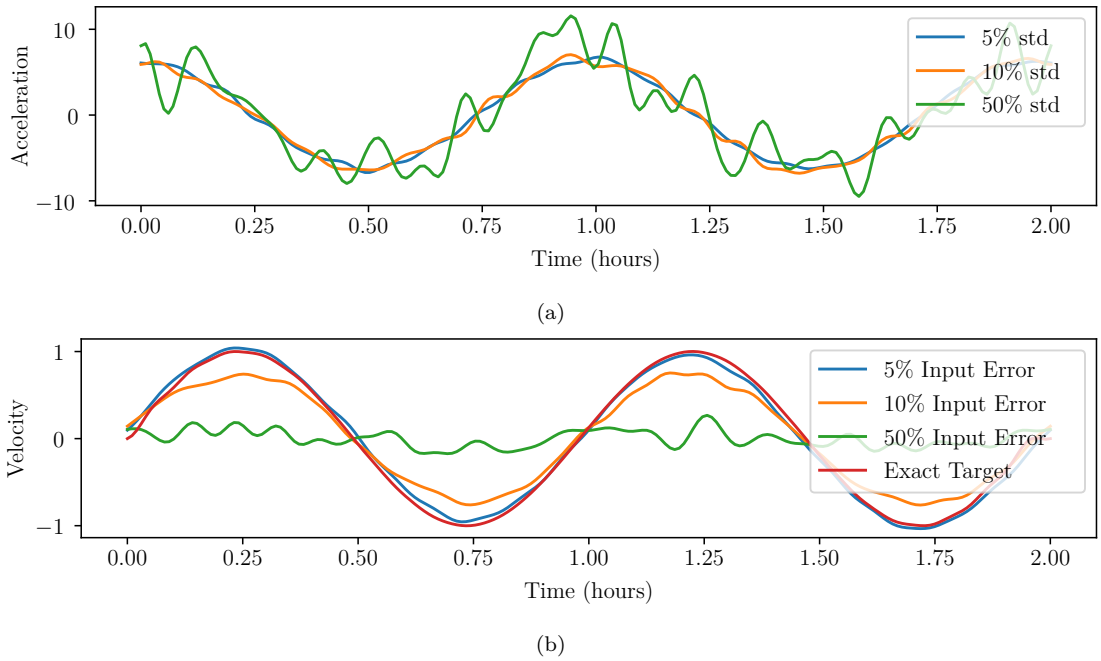


Figure 3: (a) The perturbed exact input function values from equation (10). (b) Prediction of antiderivative from the input function that was perturbed.

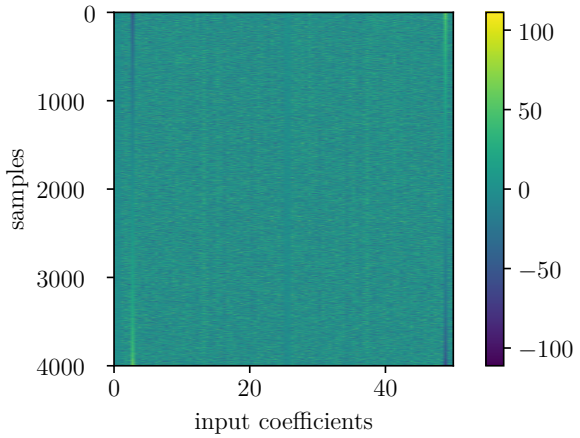
Next, we have the correlation image and p-matrix for the model of each noise level. The results are shown in figure 4. Each row in the figure represent the results of a model trained on a different noise level. The correlation image for 5% and 10% noise show clear lines on the coefficient corresponding to the output coefficient that was sorted. We can also see that the negative wave number reflection also being sorted. This is because of the reflection that occurs with complex Fourier coefficients for real-valued functions. There are other faint vertical lines you are able to see for other input coefficients. These faint lines are not sorted like the corresponding coefficients we mentioned before. Meaning, while they don't affect the particular output coefficients that were sorted, they do show that there is

information embedded in the kernel matrix for these input coefficients. However, if we look at the 50% noise correlation image, even the corresponding coefficient does not show a clear line or gradient. The lines are more noisy compared to the other noise levels. But the kernel still manages to embed some information. The p-matrices show that the model itself is able to still learn some relationship between the input coefficients and the output coefficients. For the 5% and 10% noise levels, the p-matrices show very clearly the contributions of input coefficients to the output coefficients. However, looking at the p-matrix for 50% noise level, the lower wave numbers show lower contribution of input coefficients to the output coefficients.

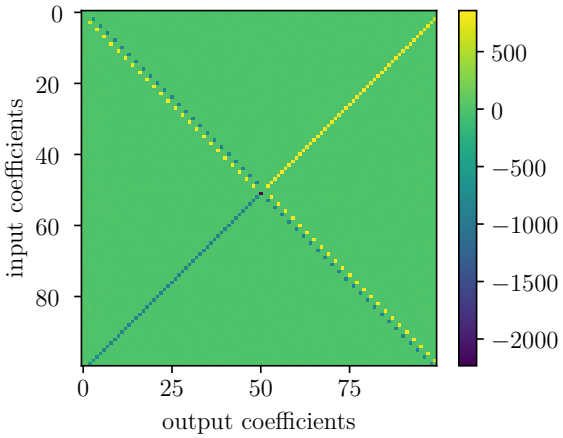
The final observation we make is how for all the p-matrices the pronounced contribution of the input wave number to their corresponding output wave number. This means that the majority of contributions to each output coefficients come from the corresponding input coefficients of the same wave number. Our knowledge on how the simple derivative equation for Fourier series relate the coefficients of the derivative function and the antiderivative function is shown in equation (5). This aligns with the contributions shown by the p-matrices. This confirms that the model is indeed learning the relations that is defined by the derivative equation.

3.2. Scenario 2: Burgers' Equation

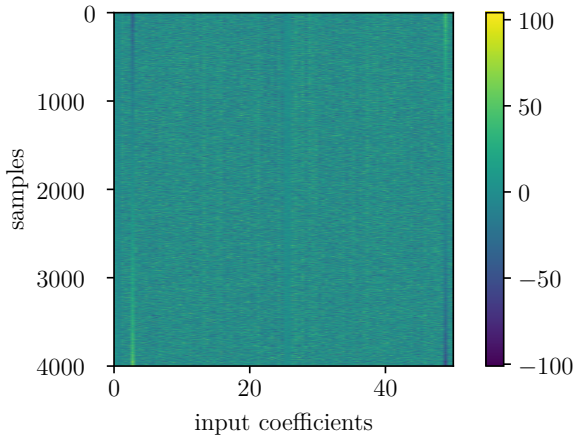
The function value evaluated from the predictions is shown in table 4. The metrics across the board shows that the function value relative metrics are slightly better compared to the coefficients. Comparing the absolute metrics for the same table, we see the same general trend that the higher viscosity show the model performing worse. For reference, the maximum amplitude of the functions on average are 2.56×10^{-2}. This puts the error in an order of magnitude less than the average maximum amplitude.



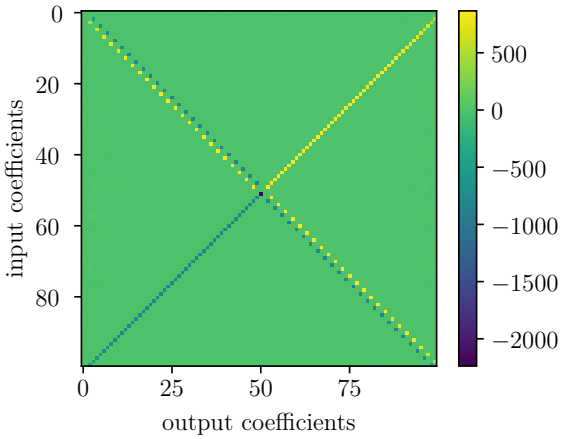
(a) Correlation image 5% noise.



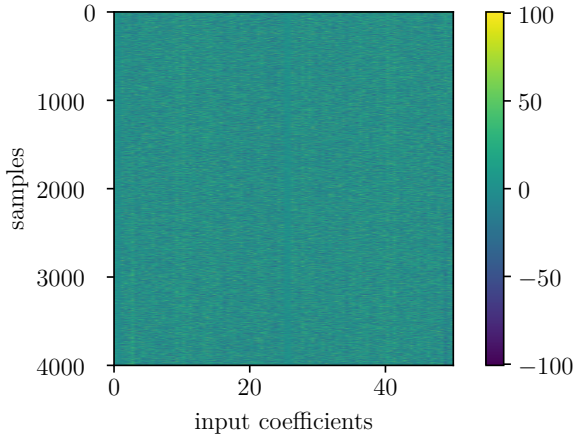
(b) The p-matrix for 5% noise.



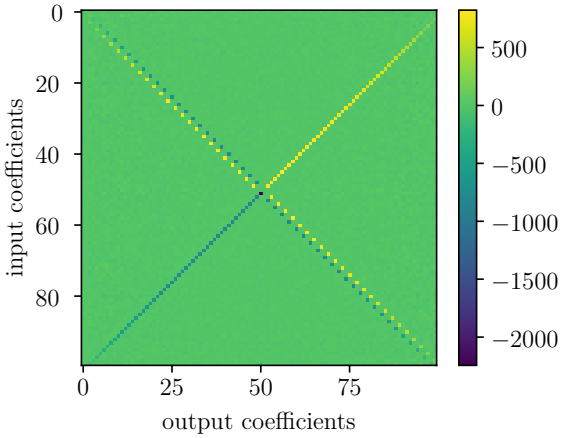
(c) Correlation image 10% noise.



(d) The p-matrix for 10% noise.



(e) Correlation image 50% noise.



(f) The p-matrix for 50% noise.

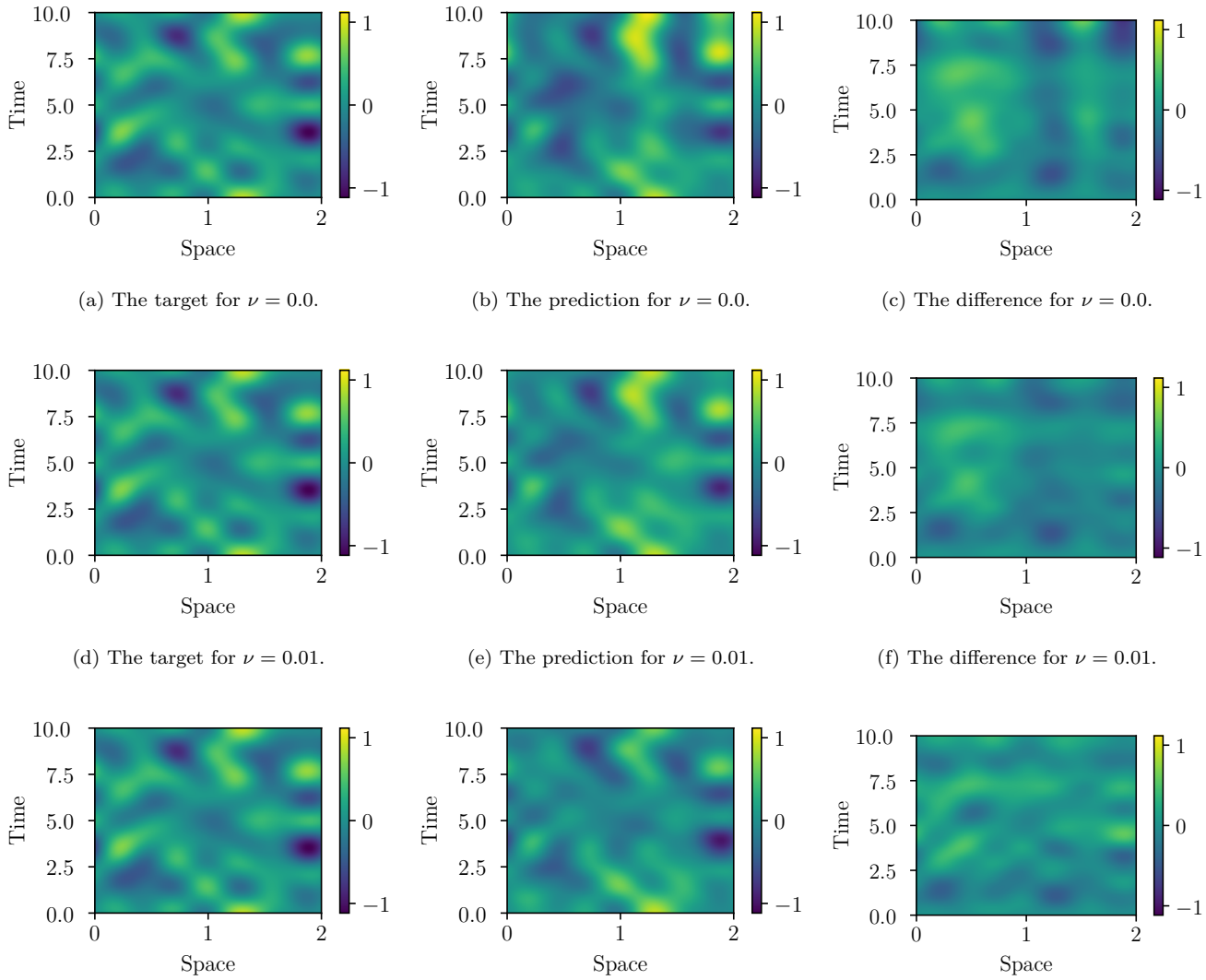
Figure 4: Correlation image (left column) and p-matrix (right column) for each model trained on a different noise level (row). The correlation image was sorted same order the values of the real component of wave number $k = 2$ were sorted in descending order.

Table 4: Performance metrics of function values evaluated from coefficient prediction of next time step in scenario 2 by viscosity.

ν	MSE	RMSE	MAE	R^2	sMAPE
0.0	5.58e-05	7.47e-03	5.89e-03	0.82	0.70
0.01	5.74e-05	7.58e-03	6.01e-03	0.81	0.71
0.1	1.58e-04	1.26e-02	9.98e-03	0.51	1.04

The testing results are complimented with a rollout experiment. The results from the rollout are shown in figure 5. The rollout starts from time $t = 0$. In all the plots, this is at the bottom. We have included this initial condition in the plots. From this initial condition in combination with the forcing term at the initial time step, the model predicts the first time step solution. The model then uses the predicted solution combined with the corresponding forcing term to predict the second time step. To see how well the model is performing relative to the actual target values, we compute the difference between the two as shown in figures 5c, 5f and 5i. Observe that errors are introduced relatively early around time $t = 1$. However, the model rollout stays stable until the end despite this error. This results in the model being able to finish the rollout for this sample.

Another observation we can see is that the difference between the target and rollout predictions is much more pronounced for lower viscosity values. This result is aligned with metrics shown in tables 5. The absolute metrics show that the models are performing several times worse in rollout compared to the single time step tests shown in tables 4. This seems to be the same challenge that traditional solvers also face with the shocks that may be present with lower viscosity values. This means that for rollout scenarios, the model has a much harder time with lower viscosity values. The observation here is that for higher viscosity values, the error is dominated by the global error, and the lower viscosity value, the error is dominated by local errors and as such accumulates.



(a) The target for $\nu = 0.0$.

(b) The prediction for $\nu = 0.0$.

(c) The difference for $\nu = 0.0$.

(d) The target for $\nu = 0.01$.

(e) The prediction for $\nu = 0.01$.

(f) The difference for $\nu = 0.01$.

(g) The target for $\nu = 0.1$.

(h) The prediction for $\nu = 0.1$.

(i) The difference for $\nu = 0.1$.

Figure 5: The rollout predictions for one of the test function. The difference is calculated as the targets subtracted by the predictions.

Table 5: Performance metrics of function values evaluated from coefficient rollout in scenario 2 by viscosity.

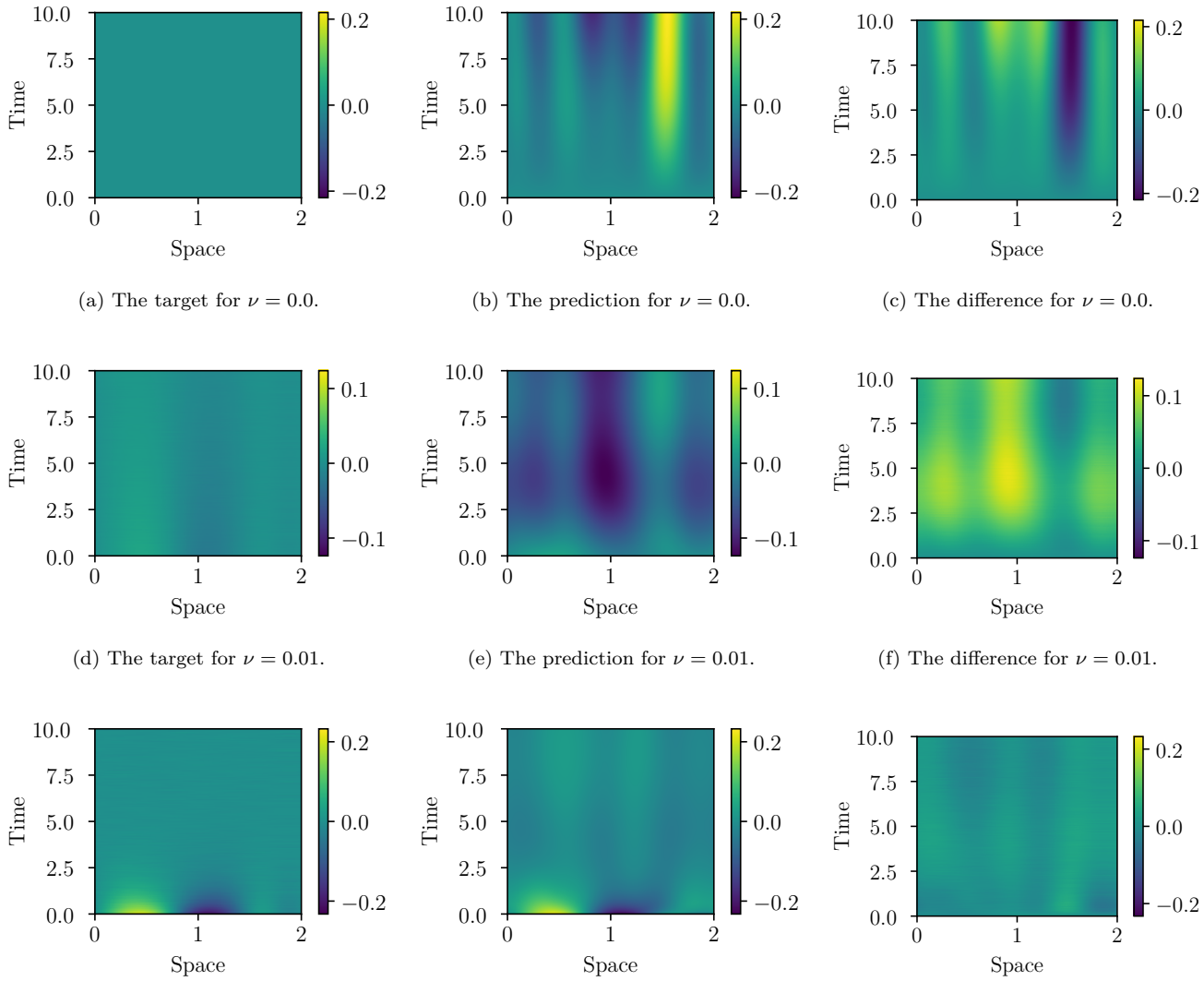
ν	MSE	RMSE	MAE	sMAPE
0.0	5.81e-02	2.41e-01	1.89e-01	1.04
0.01	3.17e-02	1.78e-01	1.40e-01	0.94
0.1	2.91e-02	1.71e-01	1.33e-01	0.92

To test the generalization capability of the model on other function families, we use an exact solution provided in the literature shown in equation (12) (Wood, 2006; Wazwaz, 2010; Benton and Platzman, 1972).

$$u(x,t) = \frac{2\nu\pi e^{-\pi^2\nu t}\sin(\pi x)}{a + e^{-\pi^2\nu t}\cos(\pi x)} \tag{12}$$

The exact function eq. (12) is computed for each viscosity value. The discrete Fourier transform of the values are then used for doing rollout with the model. For the forcing term, we just use a constant function with a value of zero. The rollout predictions are shown in figure 6. The model is not successful in the rollout for the inviscid equation and for the viscosity of $\nu = 0.01$. Both of these cases, the model produces too much error that the original function is no longer recognizable in the predictions. For the higher viscosity value of $\nu = 0.1$ the rollout produces a recognizable prediction. The error stays to about half the maximum function value. We do see some more noticeable error in the flatter parts of the function. The error also take a very similar shape across the different viscosity values. Since we essentially used the same solutions for training the model for each viscosity, the similarity in error means that it can be alleviated with more diverse training samples. The error for the flatter functions may also be alleviated by including similar flatter functions during training.

The correlation image and p-matrix for each viscosity can be seen in figure 7. Analyzing the correlation images first, we see that some columns are sorted



(a) The target for $\nu = 0.0$.

(b) The prediction for $\nu = 0.0$.

(c) The difference for $\nu = 0.0$.

(d) The target for $\nu = 0.01$.

(e) The prediction for $\nu = 0.01$.

(f) The difference for $\nu = 0.01$.

(g) The target for $\nu = 0.1$.

(h) The prediction for $\nu = 0.1$.

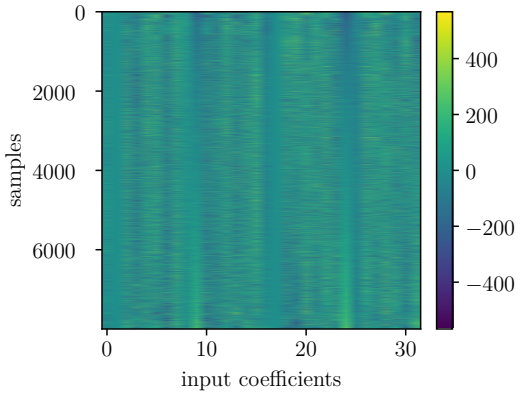
(i) The difference for $\nu = 0.1$.

Figure 6: The rollout predictions for one of the exact function in equation (12). The difference is calculated as the targets subtracted by the predictions.

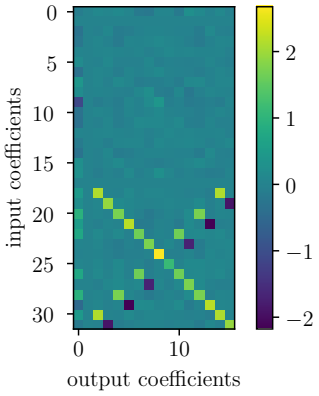
in correlation to how the samples are sorted based on the real component of the 4th output wave number. For the inviscid equation, this correlation is the strongest with multiple columns exhibiting the correlation. The highest and visually most sorted values are from the forcing term portion of the inputs. Different wave numbers in the inputs are also sorted differently, this indicates an inverse correlation. For a higher viscosity of $\nu = 0.01$, a weaker correlation is still present.

Once we get to the highest viscosity, while the order is still present, we see a more random arrangement of the correlation image values. This observation and the lower scores can both be explained by the level of temporal discretization. This is the same issue we encounter when we use some traditional methods for higher viscosity values with the same time step for lower viscosity values (Kassam and Trefethen, 2005; Seydaoğlu et al., 2016). The issue arises in stiff differential equations. Stiff differential equations refer to the presence of a rapidly varying component. We can see that the larger amplitudes of high frequency components in the forcing term to represent the rapidly varying components as shown in figures B.11b, B.11d and B.11f. Traditionally, stiff equations would require specific numerical methods purpose built to efficiently solve the problem. Otherwise, the using a solver like FDM would require much finer time steps than the one we are using here.

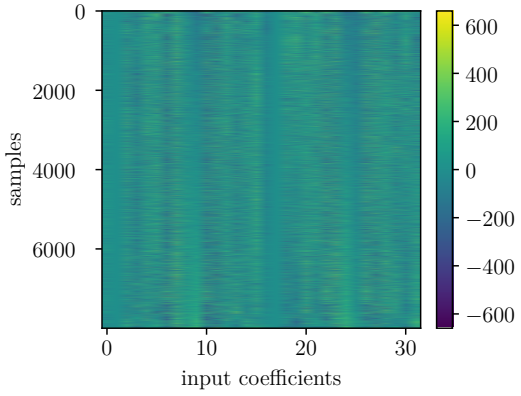
The p-matrix shows the difference in contributions of both input functions. For the inviscid equation, the contributions mainly come from the forcing term. The current solution contributes a smaller amount in comparison. There is similarity with the antiderivative p-matrices in this case. The contributions to each output wave number mainly come from input coefficients with the same wave number. However, there is a noticeable contribution from other wave numbers too unlike the antiderivative case. The biggest contribution from the current solution is the constant/bias term. For higher viscosity values, notice that the contribution of the previous solution increases. The increase is more pronounced for higher



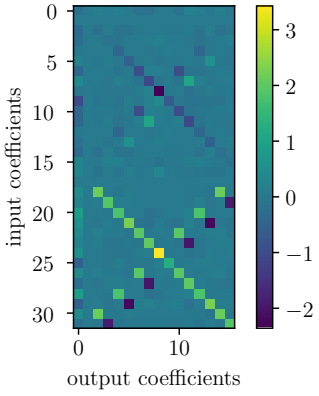
(a) Correlation image $\nu = 0.0$.



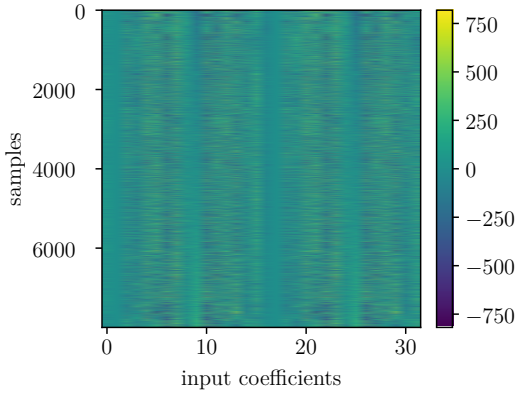
(b) The p-matrix for $\nu = 0.0$.



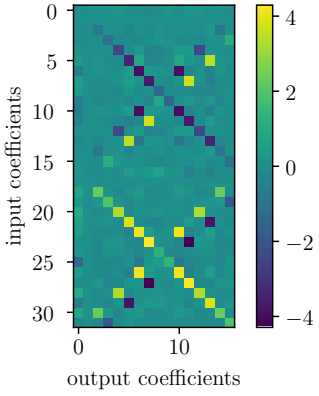
(c) Correlation image $\nu = 0.01$.



(d) The p-matrix for $\nu = 0.01$.



(e) Correlation image $\nu = 0.1$.



(f) The p-matrix for $\nu = 0.1$.

Figure 7: Correlation image (left column) and p-matrix (right column) for each model trained on a different viscosity (row). The correlation image was sorted same order the values of the real component of wave number $k = 2$ were sorted in descending order.

frequencies. This aligns with the quadratic scaling from the wave number on the coefficients (Canuto et al., 2007). The p-matrix for lower viscosity values show more structured contribution of other wave numbers in comparison to the higher wave numbers. This structure comes in the form of a grid pattern. For the real components of the output, the contribution of other wave numbers oscillate between higher and lower values depending on whether the input component is real or not. The imaginary output components on the other hand have a more constant contribution across the input components. The contribution once again become more pronounced the higher the absolute value of the wave number.

3.3. Comparisons

As a point of reference, the comparisons of our method against three other methods will be discussed for solving the Burger's equation as an initial value problem. The methods we compare against are the lsoda program using the SciPy library, the Implicit Adams-Bashforth-Moulton (IABM) method using the torchdiffeq library, and using a Fully-connected Neural Network (FNN) in place of the LSSVR to directly model the coefficient relationships. Both the lsoda and IABM approaches discretize the solution in space into 400 points and the periodicity is enforced using a ghost cell at one end that has the same value as the other end. First, we present the results of solving equation (12) as an IVP as shown in table 6 for viscosity $\nu = 0.1$. For the forcing term, we use a constant value of zero. The functions are also unperturbed. The metrics measure the performance of the predicted solutions compared against the exact solution. We see that our model has slightly better performance compared to the lsoda method. However, for lower viscosity values the lsoda method performs much better compared to ours as shown in Appendix C. And for the viscosity of $\nu = 0.0$, even the IABM method performs much better due to the fact that the solution is a constant value of zero. The baseline FNN method, however, does not perform well on any of the viscosity values with the solution becoming unstable after several time steps.

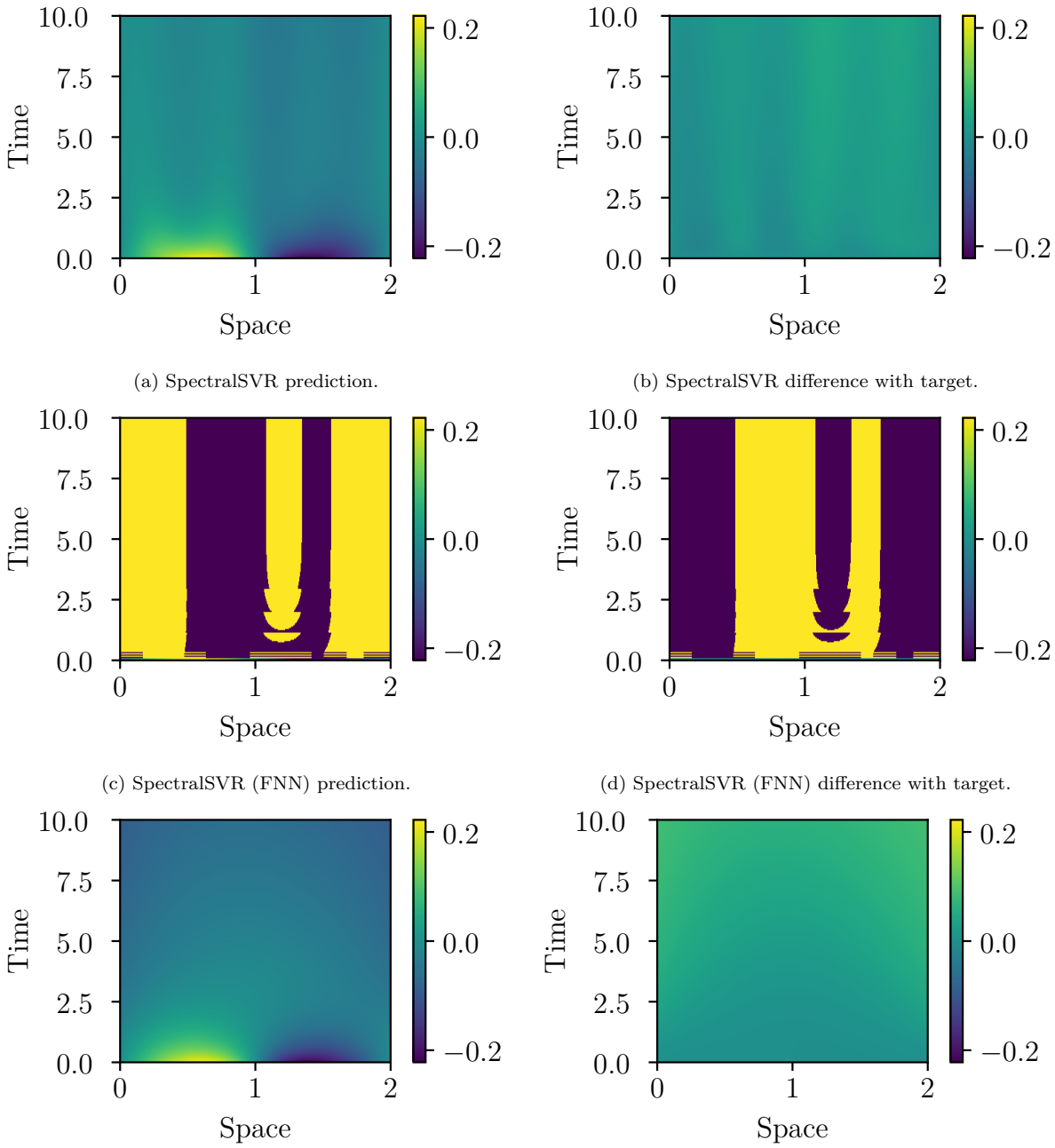
Table 6: Metrics for predicting the exact solution for each method with viscosity $\nu = 0.1$.

Method	MSE	RMSE	MAE	sMAPE
SpectralSVR	0.00	0.02	0.02	1.47
SpectralSVR (FNN)	inf	inf	inf	1.98
lsoda	0.00	0.04	0.04	1.52
IABM	nan	nan	nan	nan

For better visualization, the predictions and the difference against the exact solution is shown in figure 8. We show that for our method and the lsoda method, the exact solution is achieved to a recognizable form. However, using the IABM method with torchdiffeq is ineffective, and the solution "blows up" and results in many not a number values, which is why we chose not to display them. Our method has errors characterized by "lines" which stems form our use of only eight basis functions to represent solution. This is in contrast to the error of the lsoda method, which is spread over larger areas.

The predictions for IVP rollout of the functions in the test set is shown in figures 9 for viscosity $\nu = 0.1$. Other viscosity values were also compared and presented in figures C.12 and C.13. An obvious observation is the fact that the numerical methods fail to solve the entire solution. The SciPy odeint (lsoda) version is able to solve up until the 13th iteration or time $t = 0.6533$. The torchdiffeq (IABM) version on the other hand only solves up until the second iteration. Even at this early step, the solution is already unstable with many values having an order of larger than 10×10^{10}.

In terms of efficiency, we also present our measurements of the time it took for each method to solve the exact solution IVP using the `%timit` macro in Jupyter Notebook. This was done on our machine without any hardware accelerators such as GPUs. The time was averaged over 7 measurements. The results are shown in table 7. Our method solves this problem around 33 times faster than the



(a) SpectralSVR prediction.

(b) SpectralSVR difference with target.

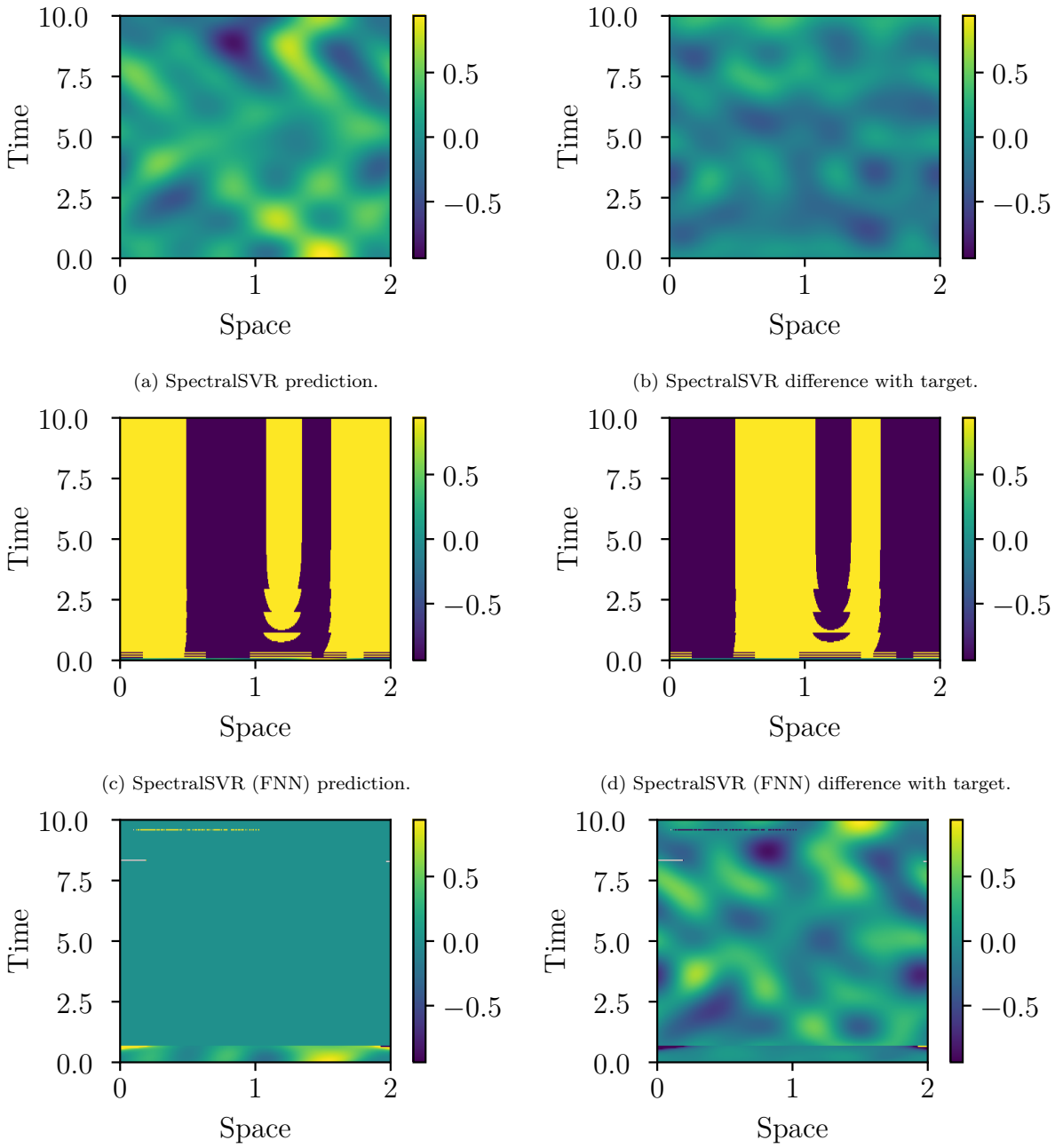
(c) SpectralSVR (FNN) prediction.

(d) SpectralSVR (FNN) difference with target.

(e) lsoda prediction.

(f) lsoda difference with target.

Figure 8: Comparison of our model (SpectralSVR), baseline FNN model, and numerical methods for the exact solution in eq. (12) of the Forced Burgers equation with viscosity $\nu = 0.1$.



(a) SpectralSVR prediction.

(b) SpectralSVR difference with target.

(c) SpectralSVR (FNN) prediction.

(d) SpectralSVR (FNN) difference with target.

(e) lsoda prediction.

(f) lsoda difference with target.

Figure 9: Comparison of our model (SpectralSVR), baseline FNN model, and numerical methods for a manufactured solution of the Forced Burgers equation with viscosity $\nu = 0.1$.

lsoda method. However, the kernel computation required does slow the LSSVR in comparison to the neural network version by around 140 milliseconds.

Table 7: Average time elapsed in solving for equation (12) by method.

Method	Time Elapsed
SpectralSVR	0.438 s ± 0.075 s
SpectralSVR (FNN)	**0.298 s ± 0.054 s**
lsoda	15.1 s ± 2.85 s
IABM	20.9 s ± 2.98 s

In terms of training, the FNN baseline model with 7 epochs and a batch size of 16, training takes on average 14.6 seconds ± 0.86 seconds. The LSSVR version takes on average 8 seconds ± 2.47 seconds.

4. Conclusion

Using the LSSVR to predict the Fourier coefficients of approximate solutions to PDEs, we demonstrate the advantages of this approach in terms of efficiency and interpretability. Because of the potential this approach has shown, it is a promising candidate for further research.

Based on the limitations of this work, we highlight three areas of interest for future research. First, our focus on the use of the Fourier basis means that it is much more suited for certain geometries. This may cause aberrations when used with unsuitable geometries such as a sphere (Bonev et al., 2023-07-23/2023-07-29). Which is why other kinds of basis functions such as spherical harmonics or even learned basis functions is of great interest in a variety of applications. Second, our configuration for generating the training data means that it is mathematically correct (Roache, 2002; Salari and Knupp, 2000; Vedovoto et al., 2011) but not representative of the desired distribution of function types; in addition to lacking realism. Because of this, an investigation into different distributions of function

types like using more realistic solutions from traditional solvers would allow the model to handle challenges like flatness. Last but not least, improving the interpretability of the model such as gaining insight into the contributions of each term in a governing equation will build upon the visualization provided by the p-matrix.

References

Basir, S., Senocak, I., 2022. Critical Investigation of Failure Modes in Physics-informed Neural Networks, in: AIAA SCITECH 2022 Forum, American Institute of Aeronautics and Astronautics, San Diego, CA & Virtual. doi:10.2514/6.2022-2353.

Benton, E.R., Platzman, G.W., 1972. A table of solutions of the one-dimensional Burgers equation. Quarterly of Applied Mathematics 30, 195–212. doi:10.1090/qam/306736.

Bonev, B., Kurth, T., Hundt, C., Pathak, J., Baust, M., Kashinath, K., Anandkumar, A., 2023-07-23/2023-07-29. Spherical Fourier neural operators: Learning stable dynamics on the sphere, in: Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., Scarlett, J. (Eds.), Proceedings of the 40th International Conference on Machine Learning, PMLR. pp. 2806–2823.

Brauer, F., Castillo-Chávez, C., Feng, Z., 2019. Mathematical Models in Epidemiology. Number volume 69 in Texts in Applied Mathematics, Springer, New York, NY.

Braun, M., 1993. Differential Equations and Their Applications: An Introduction to Applied Mathematics. Number 11 in Texts in Applied Mathematics. fourth edition ed., Springer, New York, NY. doi:10.1007/978-1-4612-4360-1.

Canuto, C., Hussaini, M.Y., Quarteroni, A., 2007. Spectral Methods: Evolution

to Complex Geometries and Applications to Fluid Dynamics. Scientific Computation, Springer e-books, Berlin Heidelberg.

Du, Y., Chalapathi, N., Krishnapriyan, A.S., 2024. Neural spectral methods: Self-supervised learning in the spectral domain, in: The Twelfth International Conference on Learning Representations.

Fanaskov, V.S., Oseledets, I.V., 2023. Spectral Neural Operators. Doklady Mathematics 108, S226–S232. doi:10.1134/S1064562423701107.

Frank, L.R., Galinsky, V.L., Zhang, Z., Ralph, F.M., 2024. Characterizing the dynamics of multi-scale global high impact weather events. Scientific Reports 14, 18942. doi:10.1038/s41598-024-67662-x.

Groetsch, C.W., 1993. Inverse Problems in the Mathematical Sciences. Vieweg+Teubner Verlag, Wiesbaden. doi:10.1007/978-3-322-99202-4.

Holmes, E.E., Lewis, M.A., Banks, J.E., Veit, R.R., 1994. Partial Differential Equations in Ecology: Spatial Interactions and Population Dynamics. Ecology 75, 17–29. doi:10.2307/1939378.

Kassam, A.K., Trefethen, L.N., 2005. Fourth-Order Time-Stepping for Stiff PDEs. SIAM Journal on Scientific Computing 26, 1214–1233. doi:10.1137/S1064827502410633.

Kopriva, D.A., 2009. Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers. Scientific Computation, Springer Netherlands, Dordrecht. doi:10.1007/978-90-481-2261-5.

Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., Mahoney, M.W., 2021. Characterizing possible failure modes in physics-informed neural networks, in: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc.. pp. 26548–26560.

Leake, C., Johnston, H., Smith, L., Mortari, D., 2019. Analytically Embedding Differential Equation Constraints into Least Squares Support Vector Machines Using the Theory of Functional Connections. Machine Learning and Knowledge Extraction 1, 1058–1083. doi:10.3390/make1040060.

Lledó, L., Thomas Haiden, Josef Schröttle, Richard Forbes, 2023. Scale-dependent verification of precipitation and cloudiness at ECMWF. https://www.ecmwf.int/en/newsletter/174/earth-system-science/scale-dependent-verification-p

Lu, L., Jin, P., Pang, G., Zhang, Z., Karniadakis, G.E., 2021. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature Machine Intelligence 3, 218–229. doi:10.1038/s42256-021-00302-5.

Mehrkanoon, S., Suykens, J.A., 2015. Learning solutions to partial differential equations using LS-SVM. Neurocomputing 159, 105–116. doi:10.1016/j.neucom.2015.02.013.

Meuris, B., Qadeer, S., Stinis, P., 2023. Machine-learning-based spectral methods for partial differential equations. Scientific Reports 13, 1739. doi:10.1038/s41598-022-26602-3.

Olver, P.J., 2014. Introduction to Partial Differential Equations. Undergraduate Texts in Mathematics, Springer International Publishing, Cham. doi:10.1007/978-3-319-02099-0.

Raissi, M., Perdikaris, P., Karniadakis, G., 2019. 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. doi:10.1016/j.jcp.2018.10.045.

Rathore, P., Lei, W., Frangella, Z., Lu, L., Udell, M., 2024. Challenges in Training PINNs: A Loss Landscape Perspective. doi:10.48550/ARXIV.2402.01868.

Roache, P.J., 2002. Code Verification by the Method of Manufactured Solutions. Journal of Fluids Engineering 124, 4–10. doi:10.1115/1.1436090.

Salari, K., Knupp, P., 2000. Code Verification by the Method of Manufactured Solutions. Technical Report SAND2000-1444, 759450. Sandia National Laboratories. doi:10.2172/759450.

Schiesser, W.E., 2012. The Numerical Method of Lines: Integration of Partial Differential Equations. Elsevier Science, s.l.

Selvadurai, A.P.S., 2000. Partial Differential Equations in Mechanics 2: The Biharmonic Equation, Poisson's Equation. Springer, Berlin Heidelberg. doi:10.1007/978-3-662-09205-7.

Seydaoğlu, M., Erdoğan, U., Öziş, T., 2016. Numerical solution of Burgers' equation with high order splitting methods. Journal of Computational and Applied Mathematics 291, 410–421. doi:10.1016/j.cam.2015.04.021.

Suykens, J.A.K., 2005. Least Squares Support Vector Machines. Repr ed., World Scientific, London.

Turchin, P., 2001. Does population ecology have general laws? Oikos 94, 17–26. doi:10.1034/j.1600-0706.2001.11310.x.

Vapnik, V.N., 2000. The Nature of Statistical Learning Theory. Statistics for Engineering and Information Science. 2nd ed ed., Springer New York, New York, NY. doi:10.1007/978-1-4757-3264-1.

Vedovoto, J.M., Silveira Neto, A.D., Mura, A., Figueira Da Silva, L.F., 2011. Application of the method of manufactured solutions to the verification of a pressure-based finite-volume numerical scheme. Computers & Fluids 51, 85–99. doi:10.1016/j.compfluid.2011.07.014.

Vogel, C.R., 2002. Computational Methods for Inverse Problems. Number 23 in Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), Philadelphia, Pa. doi:10.1137/1.9780898717570.

Wang, R., Kashinath, K., Mustafa, M., Albert, A., Yu, R., 2020. Towards Physics-informed Deep Learning for Turbulent Flow Prediction, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, Virtual Event CA USA. pp. 1457–1466. doi:10.1145/3394486.3403198.

Wazwaz, A.M., 2010. Partial Differential Equations and Solitary Waves Theory. Nonlinear Physical Science, Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-642-00251-9.

Wood, W.L., 2006. An exact solution for Burger's equation. Communications in Numerical Methods in Engineering 22, 797–798. doi:10.1002/cnm.850.

Youxi Wu, Ying Li, Lei Guo, Weili Yan, Xueqin Shen, Kun Fu, 2005. SVM for Solving Forward Problems of EIT, in: 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference, IEEE, Shanghai, China. pp. 1559–1562. doi:10.1109/IEMBS.2005.1616732.

Appendix A. Derivative Equation Generated Data Example

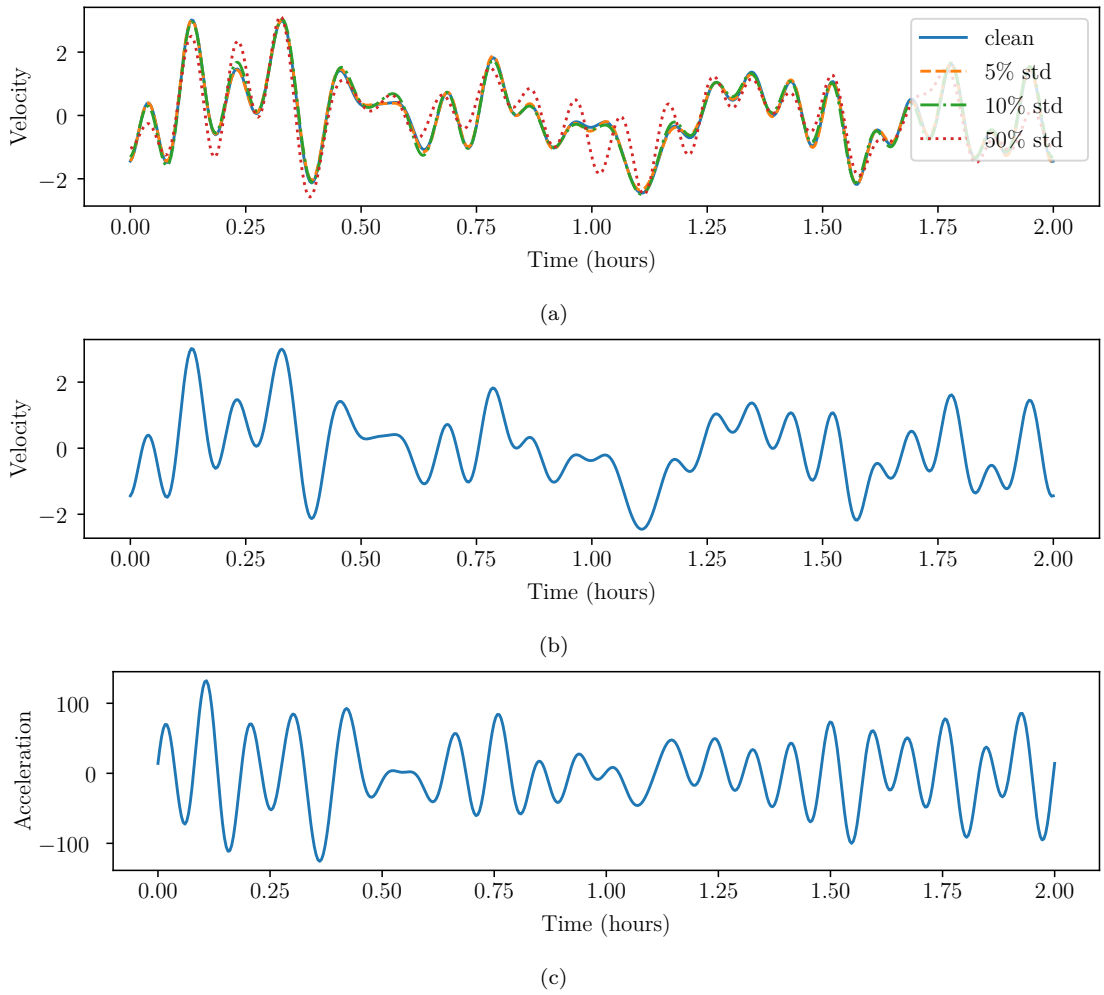
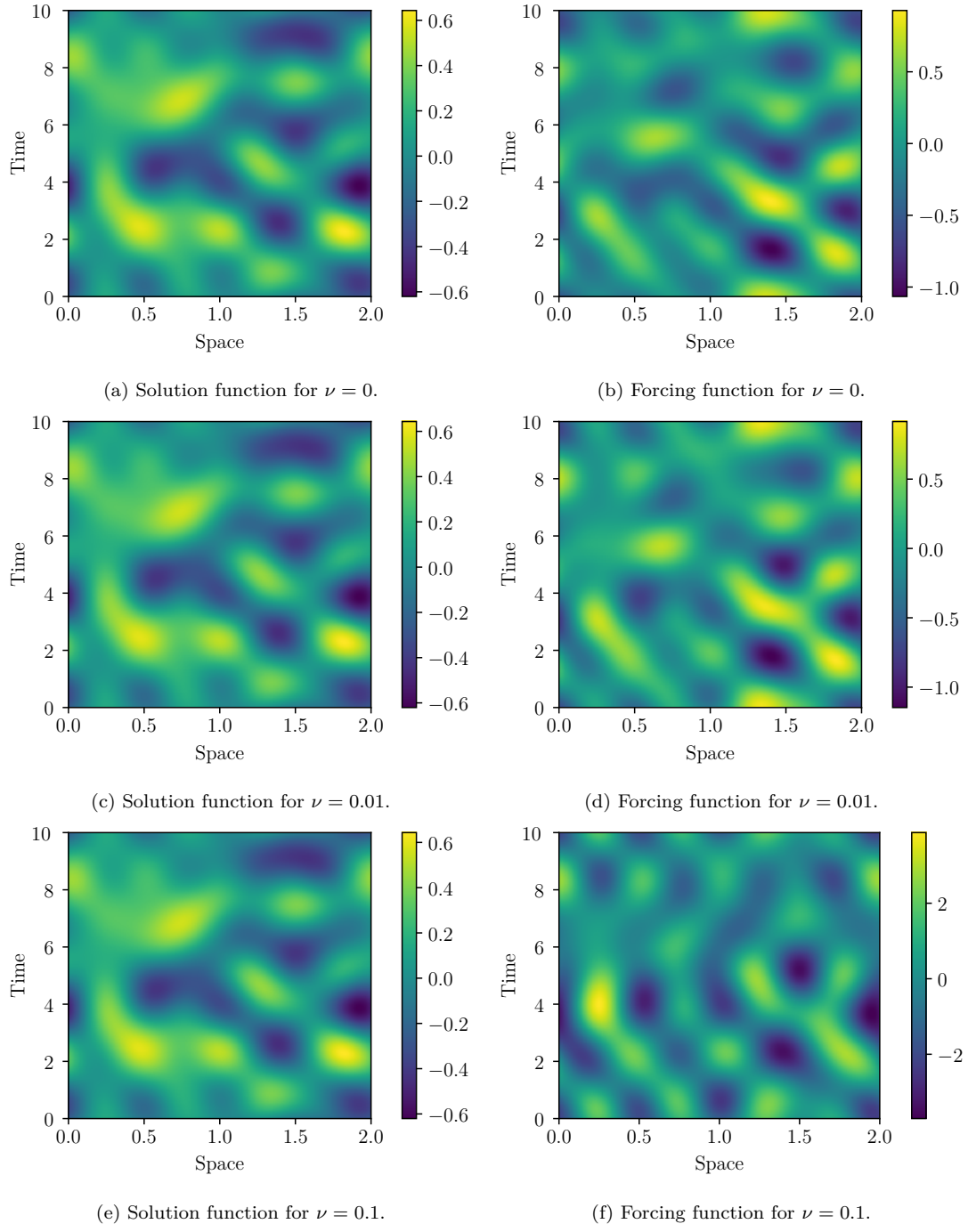


Figure A.10: (a) A generated function with no noise (clean), low noise level (5%), medium noise level (10%), and high noise levels (50%). (b) Antiderivative function with no noise. (c) Derivative function with no noise.

Appendix B. Burgers' Equation Generated Data Example



(a) Solution function for $\nu = 0$.

(b) Forcing function for $\nu = 0$.

(c) Solution function for $\nu = 0.01$.

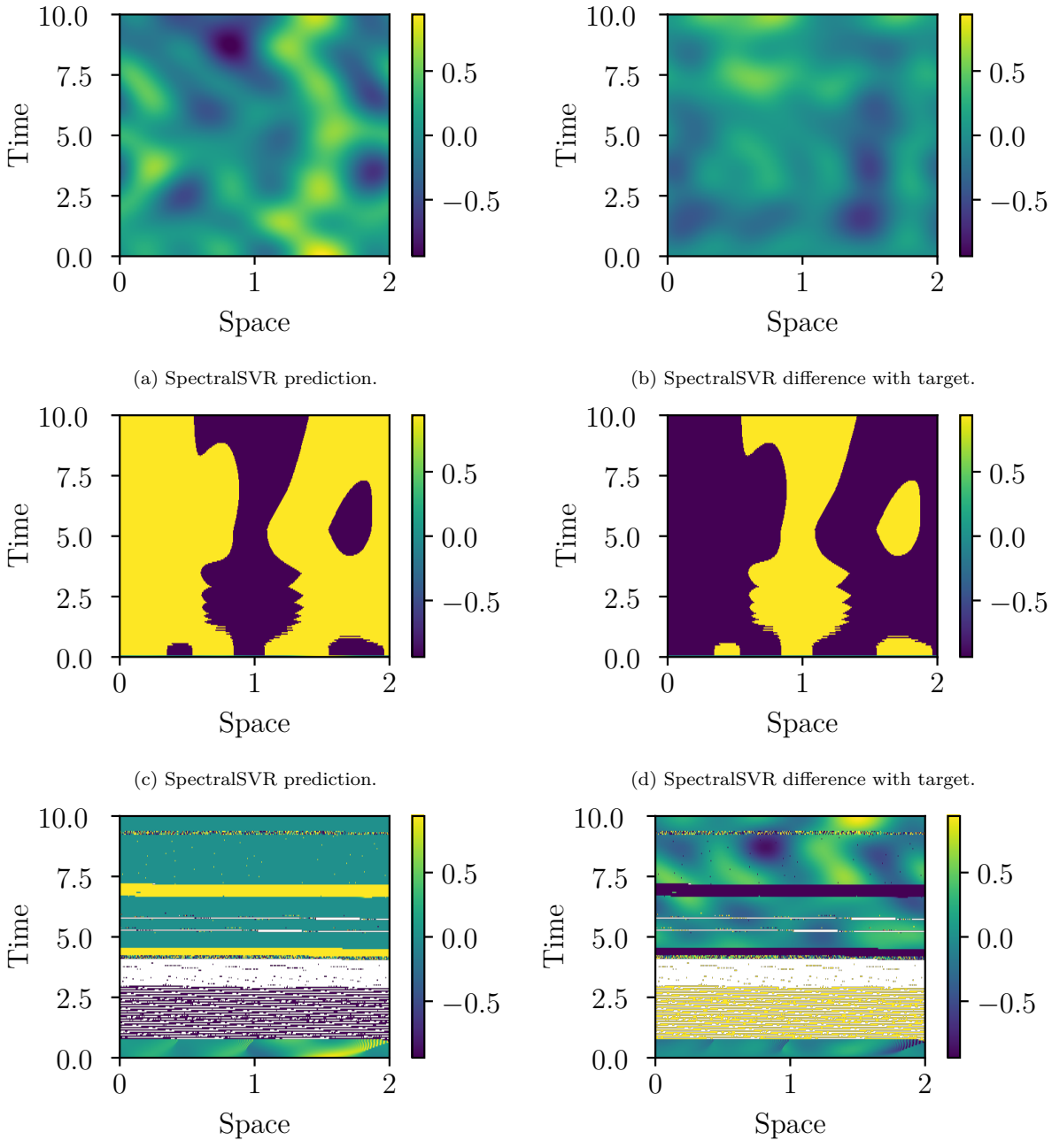
(d) Forcing function for $\nu = 0.01$.

(e) Solution function for $\nu = 0.1$.

(f) Forcing function for $\nu = 0.1$.

Figure B.11: Example sample pairs of solution and forcing terms for the Burgers' equation dataset.

Appendix C. Burgers' Equation Comparison



(a) SpectralSVR prediction.

(b) SpectralSVR difference with target.

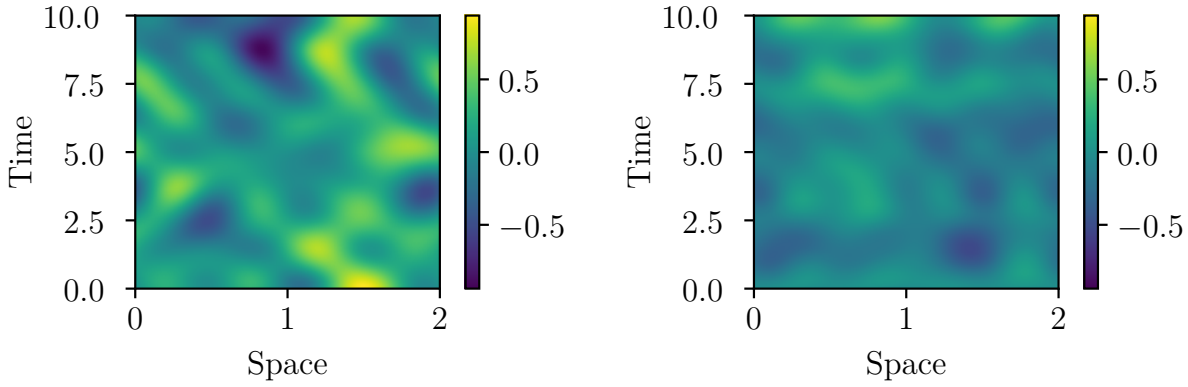
(c) SpectralSVR prediction.

(d) SpectralSVR difference with target.

(e) lsoda prediction.

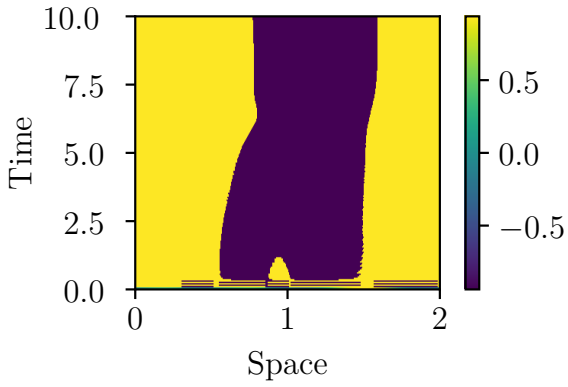
(f) lsoda difference with target.

Figure C.12: Comparison of our model (SpectralSVR) and numerical methods for the Forced Burgers equation with viscosity $\nu = 0.0$.

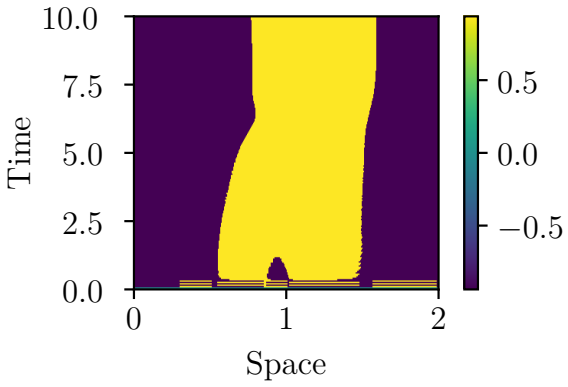


(a) SpectralSVR prediction.

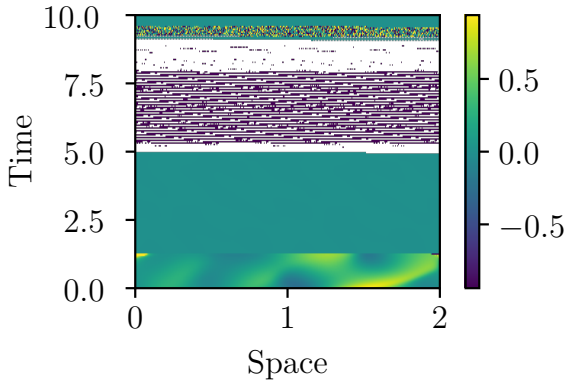
(b) SpectralSVR difference with target.



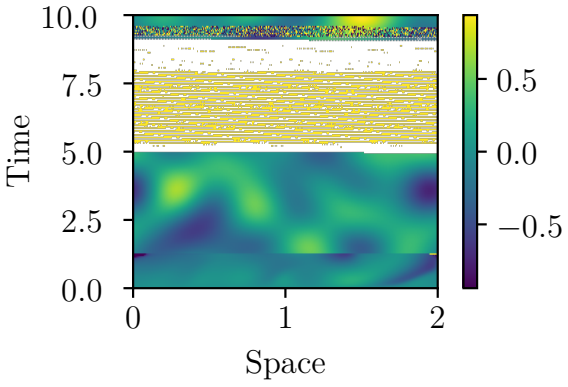
(c) SpectralSVR prediction.



(d) SpectralSVR difference with target.



(e) lsoda prediction.



(f) lsoda difference with target.

Figure C.13: Comparison of our model (SpectralSVR) and numerical methods for the Forced Burgers equation with viscosity $\nu = 0.01$.

Table C.8: Metrics for predicting the exact solution for SpectralSVR (Ours) method.

ν	MSE	RMSE	MAE	sMAPE
0.0	0.01	0.07	0.06	1.99
0.01	0.00	0.07	0.06	1.61
0.1	0.00	0.02	0.02	1.47

Table C.9: Metrics for predicting the exact solution for SpectralSVR (FNN) baseline method.

ν	MSE	RMSE	MAE	sMAPE
0.0	inf	inf	inf	1.99
0.01	inf	inf	inf	1.98
0.1	inf	inf	inf	1.98

Table C.10: Metrics for predicting the exact solution for the lsoda method.

ν	MSE	RMSE	MAE	sMAPE
0.0	0.00	0.00	0.00	0.00
0.01	0.00	0.00	0.00	0.08
0.1	0.00	0.04	0.04	1.52

Table C.11: Metrics for predicting the exact solution for the IABM method.

ν	MSE	RMSE	MAE	sMAPE
0.0	0.00	0.00	0.00	0.00
0.01	nan	nan	nan	nan
0.1	nan	nan	nan	nan