# ABSTRACT

Title of Thesis: MACHINE LEARNING-BASED
TROUBLED-CELL INDICATORS
FOR RKDG METHODS

Isabelle Stepler
Master's of Science, 2025

Thesis Directed by: Assistant Professor Deep Ray
Department of Mathematics

Solving partial differential equations (PDEs) numerically is an ongoing challenge, especially given the complicated PDEs that arise from scientific applications. Hyperbolic conservation laws are a specific type of PDEs arising from physical situations where a quantity such as mass, momentum, or energy is conserved in a fixed volume. However, the solutions to this class of PDEs often develop discontinuities as time evolves. These discontinuities often cause spurious oscillations in the numerical solution, reducing the solver's accuracy. To eliminate spurious oscillations, shock-capturing methods identify the location of discontinuities, labeling them as troubled cells, and smooth the solution in those cells. For this thesis, troubled-cell indicators are examined in the context of the Runge-Kutta Discontinuous Galerkin method for hyperbolic conservation laws. Unfortunately, many existing troubled-cell indicators rely on problem-dependent parameters that do not generalize across different initial conditions, conservation laws, or degrees of the solution. Therefore, the goal of this thesis is to compare the performance of machine-learning based methods, which are free of problem-dependent parameters, to a selection of existing troubled-cell indicators in a variety of one-dimensional cases. This thesis will discuss the use of support vector machines (SVMs) and decision trees as alternatives to traditional troubled-cell indicators and neural networks (created by Ray and Hesthaven, for example). While neural networks have been successful, their complicated nature inhibits interpretation of the troubled cell decision function. We show that SVMs are competitive with other troubled cell indicators in a variety of conservation law examples and analyze the SVM in comparison to the neural networks of Ray and Hesthaven.

# MACHINE LEARNING-BASED TROUBLED-CELL INDICATORS FOR RKDG METHODS

by

Isabelle Stepler

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master's of Science
2025

Advisory Committee:
Assistant Professor Deep Ray, Chair/Advisor
Professor James Baeder
Professor Maria Cameron

# Acknowledgments

First and foremost, I would like to thank my advisor, Dr. Deep Ray, for his unwavering support and guidance. I have learned so much and gained more confidence in my research abilities thanks to him over the course of the past two years, so I am very grateful to be his student. I would also like to express my gratitude to my personal support network for encouraging me and believing in me throughout this research project.

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| CFL | Courant-Friedrichs-Levy |
| DG | Discontinuous Galerkin |
| KXRCF | Krivonodova, Xin, Remacle, Chevaugeon, and Flaherty indicator |
| ML | Machine Learning |
| MLP | Multi-Layer Perceptron |
| ODE | Ordinary Differential Equation |
| PDE | Partial Differential Equation |
| RBF | Radial Basis Function |
| RK | Runge-Kutta |
| RKDG | Runge-Kutta Discontinuous Galerkin |
| SVM | Support Vector Machine |
| TVB | Total Variation Bounded |
| TVDM | Total Variation Diminishing in the Mean |
| WENO | Weighted Essentially Non-oscillatory |

# Chapter 1: Introduction

Scientific machine learning is a field of mathematics combining scientific computation and machine learning. Often, traditional numerical methods may have a small computational bottleneck, where the flow of code slows down and may be dependent on parameters. These bottlenecks may also be places where the numerical methods create inaccuracies in the solutions or fail altogether. Thus, scientific machine learning replaces the methods in these computational bottlenecks with a machine learning method that is just as accurate and free of problem-dependent parameters. While improving the bottleneck, this replacement also preserves the integrity of the method. In this thesis, we apply scientific machine learning to conservation law problems.

Conservation of mass, momentum, and energy are foundational concepts in physics and can be mathematically modeled by a class of partial differential equations (PDEs) called hyperbolic conservation laws. These PDEs are useful for physical problems involving flows, such as modeling tsunami propagation, tidal flows, or the development of a landslide with the shallow water equations. Additionally, the Euler equations, which relate a fluid's density, pressure, and velocity, can model the aerodynamic flow around an airfoil and its creation of lift. The magnetic nature of plasmas in the sun and in nuclear reactors can be studied using the magnetohydrodynamics equations.

All of these applications represent situations where a certain quantity is conserved in a fixed volume. In other words, conservation means that the quantity's change is determined by the amount flowing into and out of the fixed volume domain, or the flux. In mathematical terms, we represent conservation laws in two ways: the integral form,

$$\int_{\Omega_x}\int_{\Omega_t} \frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}f(u(x,t))\,dt\,dx = 0, \tag{1.1}$$

where $\Omega_x$ and $\Omega_t$ are the spatial and time domains, respectively, and $f(u)$ is the flux; or the differential form,

$$\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}f(u(x,t)) = 0. \tag{1.2}$$

However, the structure of conservation laws allow spontaneous formation of discontinuities as time moves forward, even when the initial condition is smooth. Thus, since the solution for $u$ may not be fully differentiable, we must define a weak solution for $u$ using the integral form.

**Definition 1.0.1.** *Define a smooth test function $\phi(x,t)$ such that $\phi$ is first-order differentiable in both $t$ and $x$ and is compactly supported. Then a solution $u$ and $\phi$ should satisfy*

$$\int_{\mathbb{R}}\int_{\mathbb{R}^+}\left(\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}f(u(x,t))\right)\phi(x,t)\,dt\,dx = 0. \tag{1.3}$$

*After integrating equation (1.3) by parts, we define a weak solution $u$ for the conservation law as a function that satisfies*

$$\int_{\mathbb{R}}\int_{\mathbb{R}^+}\left(u\frac{\partial}{\partial t}\phi(x,t) + f(u)\frac{\partial}{\partial x}\phi(x,t)\right)dt\,dx = -\int_{\mathbb{R}}\phi(x,0)u(x,0)\,dx, \tag{1.4}$$

*for all compactly supported test functions $\phi$.*

**Remark 1.0.1.** *There are infinitely many weak solutions $u$ to the conservation law satisfying* *equation* (1.4). *To select the physically correct solution, additional entropy conditions, like the Rankine-Hugoniot and Lax entropy conditions, are necessary. See [6] for more details.*

There are several types of numerical methods to solve hyperbolic conservation laws that try to resolve the discontinuities in different ways. Finite difference schemes use finite difference approximations for the derivative, sometimes in conjunction with artificial viscosity to reduce oscillations around the shock [20]. Finite volume methods address discontinuities by using the integral form of the conservation law, breaking the domain into cells over which the integral of the solution is approximated. More on these methods can be found in LeVeque's book, *Finite Volume Methods for Hyperbolic Problems* [10]. Discontinuous Galerkin schemes, developed from finite element methods, seek a *discontinuous* solution by imposing the weak form of the solution on individual cells, allowing them to handle complex geometries while remaining conservative and high-order accurate [1]. There also exist spectral methods for solving conservation laws [5]. However, these high-order methods are challenging to use for the discontinuous solutions due to the development of spurious Gibbs oscillations around the discontinuities. The oscillations' reduction in solution accuracy thus motivates the need for shock-capturing techniques to detect and control them.

This thesis will focus on Runge-Kutta Discontinuous Galerkin (RKDG) methods, which use a discontinuous Galerkin discretization in space and a Runge-Kutta time integration method, integrated with a shock-capturing method to limit spurious oscillations. In the RKDG context, a popular shock-capturing strategy involves first detecting the cells where discontinuities occur, called troubled cells, then correcting the numerical solution in these flagged troubled cells by either limiting the degree of the solution or adding artificial viscosity. However, these shock-capturing methods can prove computationally costly, so it is best to have an accurate troubled cell indicator that captures only the discontinuities, not cells where the solution is smooth.

Many troubled cell indicators exist, from the minmod and minmod-TVB indicators of Cockburn and Shu [1,2] to the newer Fu and Shu [4] indicator. However, many of these troubled cell indicators depend on problem-dependent parameters. These parameters can be hard to determine a priori and are typically estimated through repeated numerical experiments, which can become expensive. In addition, once the initial condition or conservation law is changed, the same parameters may not work. And, since the number of troubled cells indicated can depend on this parameter as well, parameter values that allow more troubled cells then increase the cost of the slope limiting process. Therefore, it is ideal to use a method without such problem-dependent parameters. Recently, machine learning (ML) methods like artificial neural networks [15, 16] have been used in both one-dimensional and two-dimensional problems with much success.

While neural networks are working well as troubled cell indicators, it can be difficult to interpret the actual detection method for the troubled cells. Therefore, the goal of this thesis is to explore new troubled cell indicators using support vector machines (SVMs) and decision trees, hopefully leading to insights into the prediction of troubled cells and the properties of training sets. We compare these new machine learning indicators to the neural networks and a handful of popular, well-reviewed [7, 13] troubled cell indicators to determine which machine learning method works best versus the other ML methods and versus the non-machine learning methods. We find that SVMs show much potential as a troubled cell indicator and are competitive with neural networks and traditional indicators in several cases. However, decision trees did not perform well and are thus not considered a viable method for troubled cell indicators.

The outline of the rest of this thesis is as follows. In Chapter 2, we review the Runge-Kutta

Discontinuous Galerkin method formulation used to solve conservation laws and provide details about our specific implementation. Chapter 3 describes the machine learning methods that will be newly applied to troubled cell indication, including the mathematical formulations of SVMs and decision trees, the structure of the training set, and the training process of each model in Python. In Chapter 4 we discuss both the existing troubled cell indication methods for our comparison and the new machine learning methods. Chapter 5 displays the numerical results from testing all indicators from Chapter 4 on common one-dimensional conservation laws, from linear advection to Burgers' equation and then the Euler equations. We compare solutions and troubled-cell plots, and we also analyze the sensitivity and support vectors of the SVM. And finally, Chapter 6 draws conclusions about the machine learning methods and troubled cell indicators from the results in Chapter 5 and discusses future ideas based on the findings of this thesis.

# Chapter 2: RKDG Schemes in One Dimension

Let us recall the scalar conservation law in one dimension defined in Chapter 1:

$$\frac{\partial}{\partial x}u(x,t) + \frac{\partial}{\partial x}f(u(x,t)) = 0, \quad x \in \Omega, t \in [0,T], \tag{2.1}$$

$$u(x,0) = u_0(x).$$

where $\Omega = [a,b]$ and $f(u)$ is the flux function. In addition, the conservation law may have boundary conditions describing the solution at $a$ and $b$, such as periodic or Dirichlet boundary conditions. In this thesis, we use Runge-Kutta discontinuous Galerkin (RKDG) methods, which first discretize the PDE over its spatial domain using a discontinuous Galerkin (DG) finite element method, that incorporating numerical flux and handling complex boundary conditions easily. This spatial discretization results in a system of ordinary differential equations (ODEs) on each discretized cell [2]. Then, this system of ODEs is discretized in time and solved using an explicit, high-order Runge-Kutta (RK) method [2]. During the RK time-integration steps, a slope limiter is applied to ensure that any evolving discontinuities are smoothed away (discussed more in Chapter 4). These slope limiters are essential to ensuring stability for RKDG methods and controlling spurious oscillations, while the DG and Runge-Kutta parts of the method contribute to the high-order accuracy of the method [2]. Other advantages of RKDG methods include their ability to handle complex geometries and the independence of the solutions on each cell due to the method's highly localized nature [2].

We begin by describing the polynomial bases that discretize the spatial domain and thus approximate the solution to (2.1).

## 2.1 Spatial Discretization and Polynomial Bases

We partition $\Omega$ into $K$ disjoint cells such that

$$\Omega = \bigcup_{k=1}^{K} D_k, \quad \text{where } D_k = [x_{k-\frac{1}{2}}, x_{k+\frac{1}{2}}], \tag{2.2}$$

and $h_k = x_{k+\frac{1}{2}} - x_{k-\frac{1}{2}}$ is the size of cell $D_k$. The center of cell $D_k$ is $x_k = \frac{1}{2}(x_{k-\frac{1}{2}} + x_{k+\frac{1}{2}})$. Then, we define the space of broken polynomials

$$V_h^N = \{u_h \in L_2[a,b] \ : \ u_h|_{D_k} = u_h^k \in P^N(D_k), \ 1 \leq k \leq K\}, \tag{2.3}$$

where $P^N(D_k)$ is the space of polynomials on $D_k$ with degree at most N. Thus, any $u_h \in V_h^N$ is represented as a direct sum of polynomials of (at most) degree $N$ over all cells, or

$$u_h(x) = \bigoplus_{k=1}^{K} u_h^k(x).$$

Note that $u_h^k$ can be discontinuous across cells. We approximate the solution to (2.1), $u$, as some $u_h \in V_h^N$.

### 2.1.1 Lagrange Polynomials

Since $u_h^k \in P^N(D_k)$, it can be expressed uniquely using the Lagrange polynomial basis $\{\ell_i^k(x)\}_{i=1}^{N+1} \subset P^N(D_k)$ as

$$u_h^k(x) = \sum_{i=1}^{N+1} u_i^k \ell_i^k(x), \tag{2.4}$$

where $x \in D_k$, $\{x_i^k\}_{i=1}^{N+1} \subset D_k$ are an appropriate choice of interpolating nodes, and $u_i^k = u_h^k(x_i^k)$. Recall that the Lagrange polynomials are defined as

$$\ell_i^k(x) = \prod_{j=1, j\neq i}^{N+1} \frac{x - x_j^k}{x_i^k - x_j^k}, \quad \ell_i^k(x_j^k) = \delta_{ij}. \tag{2.5}$$

Because of the second property in equation (2.5), $u_i^k \ell_i^k(x_i^k) = u_i^k$, so the Lagrangian coefficients $u_i^k$ are called the nodal coefficients [6].

### 2.1.2 The Reference Element

It is more convenient to work with the interval $D = [-1, 1]$, called the reference element, than each $D_k$. Creating the following transformation between $D_k$ and $D$ allows all polynomials, integrals, and matrices necessary for the DG scheme to be computed on one domain instead of $K$ different cells [6]. Any $r \in D$ can be mapped to $x \in D_k$ by the transformation

$$x = \mathcal{T}_k(r) = \frac{h_k}{2} r + \frac{x_{k+\frac{1}{2}} + x_{k-\frac{1}{2}}}{2}, \tag{2.6}$$

which implies that the inverse mapping from $D_k$ to $D$ is

$$r = \mathcal{T}_k^{-1}(x) = \frac{2}{h_k}x - \frac{x_{k+\frac{1}{2}} + x_{k-\frac{1}{2}}}{h_k}. \tag{2.7}$$

Knowing (2.6), now we only need to choose one set of nodes $\{r_i\}_{i=1}^{N+1}$ in $D$, independent of $k$, and define the interpolation nodes in $D_k$ as

$$x_i^k = \mathcal{T}_k(r_i), \quad 1 \leq i \leq N+1.$$

This transformation allows us to transform many calculations to the reference element and simplify the computation of integrals and coefficients, for example. Specifically, using equation (2.6), we have

$$\ell_i^k(x) = \prod_{j=1, j\neq i}^{N+1} \frac{x - x_j^k}{x_i^k - x_j^k} = \prod_{j=1, j\neq i}^{N+1} \frac{\mathcal{T}_k(r) - \mathcal{T}_k(r_j)}{\mathcal{T}_k(r_i) - \mathcal{T}_k(r_j)} = \prod_{j=1, j\neq i}^{N+1} \frac{r - r_j}{r_i - r_j} = \ell_i(r).$$

Thus, the $\ell_i(r)$ polynomials are a Lagrange interpolation basis on $D$, so we can more easily express $u_h^k$ as

$$u_h^k(x) = \sum_{i=1}^{N+1} u_i^k \ell_i(r). \tag{2.8}$$

### 2.1.3 The Modal Coefficients

We can also pick a different, orthonormal basis $\{\phi_i(r)\}_{i=1}^{N+1}$ that spans $P^N(D)$ [6] and has the property

$$\int_D \phi_i(r)\phi_j(r)\, dr = \delta_{ij}.$$

Thus, we can find modal coefficients $\{\hat{u}_i^k\}_{i=1}^{N+1}$ such that

$$u_h^k(x) = u_h^k(\mathcal{T}_k(r)) = \sum_{j=1}^{N+1} u_j^k \ell_j(r) = \sum_{j=1}^{N+1} \hat{u}_j^k \phi_j(r). \tag{2.9}$$

Using (2.9) and plugging in a specific $r_i$, we get

$$u_h^k(\mathcal{T}_k(r_i)) = \sum_{j=1}^{N+1} u_j^k \ell_j(r_i) = u_i^k = \sum_{j=1}^{N+1} \hat{u}_j^k \phi_j(r_i). \tag{2.10}$$

Therefore, by defining the vectors of nodal and modal coefficients as $\boldsymbol{u}^k = [u_1^k, \ldots, u_{N+1}^k]^T$ and $\hat{\boldsymbol{u}}^k = [\hat{u}_1^k, \ldots, \hat{u}_{N+1}^k]^T$, and defining the Vandermonde matrix $V \in \mathbb{R}^{(N+1)\times(N+1)}$ as $V_{ij} = \phi_j(r_i)$, equation (2.10) gives us the relationship

$$\boldsymbol{u}^k = V\hat{\boldsymbol{u}}^k, \quad \text{and} \quad V^{-1}\boldsymbol{u}^k = \hat{\boldsymbol{u}}^k. \tag{2.11}$$

The Vandermonde matrix is the key to transforming the nodal coefficients to the modal coefficients, and vice versa [6]. It is also used in the relationship between the two polynomial bases; since the polynomial $\phi_i(r)$ is a member of $P^N(D)$, we can write it in terms of the Lagrangian basis as

$$\phi_i(r) = \sum_{j=1}^{N+1} \phi_i(r_j)\ell_j(r), \quad \forall i = 1, \ldots, N+1. \tag{2.12}$$

Therefore, in vectorized form, with $\boldsymbol{\ell}(r) = [\ell_1(r), \ldots, \ell_{N+1}(r)]^T$ and $\boldsymbol{\phi}(r) = [\phi_1(r), \ldots, \phi_{N+1}(r)]^T$, equation (2.12) becomes

$$V^T\boldsymbol{\ell}(r) = \boldsymbol{\phi}(r). \tag{2.13}$$

## 2.2 DG Formulation

Let the global solution $u(x,t)$ to the PDE be approximated by the piecewise polynomial approximation (e.g., the direct sum of the local polynomial solutions):

$$u(x,t) \approx u_h(x,t) = \bigoplus_{k=1}^{K} u_h^k(x,t). \tag{2.14}$$

Using the Lagrange basis, we rewrite $u_h^k(x,t)$ for $x \in D_k$ as

$$u_h^k(x,t) = \sum_{i=1}^{N+1} u_i^k(t)\ell_i^k(x), \tag{2.15}$$

where $u_i^k(t) := u_h^k(x_i^k, t)$. We also can write an approximation for the flux $f$ for $x \in D_k$ using the Lagrange polynomials:

$$f(x,t) \approx f_h^k(x,t) = \sum_{i=1}^{N+1} f_i^k(t)\ell_i^k(x), \tag{2.16}$$

where $f_i^k(t) := f(u_h^k(x_i^k, t))$.

Then, to get the local solution to the PDE on cell $D_k$, we plug in equations (2.15) and (2.16) to the PDE (2.1) to form the residual on $D_k$,

$$\mathcal{R}_h^k(x,t) = \frac{\partial}{\partial t}u_h^k(x,t) + \frac{\partial}{\partial x}f_h^k(x,t) \quad \forall\, 1 \le k \le K,\ x \in D_k. \tag{2.17}$$

Recall from Chapter 1 that DG schemes are based on the weak formulation of the solution $u$ (see Definition 1.0.1 and equation (1.4)). We choose the space of test functions to be the same as the solution space on each cell: $P^N(D_k)$. Therefore, we can formulate the DG scheme by assuming that the residual $\mathcal{R}_h^k(x,t)$ is orthogonal to all functions in $P^N(D_k) = \text{span}\{\ell_i(D_k)\}_{i=1}^{N+1}$, similar to equation (1.3). In other words,

$$\int_{D_k} \mathcal{R}_h^k(x,t)\ell_i^k(x)\,dx = 0, \quad \forall\, i = 1,\ldots,N+1. \tag{2.18}$$

Substituting in the Lagrangian basis representations for $u_h^k(x,t)$ and $f_h^k(x,t)$ and integrating (2.18) by parts over $D_k = [x_{k-\frac{1}{2}}, x_{k+\frac{1}{2}}]$ for each $i = 1,\ldots,N+1$ leads to

$$\begin{aligned}\sum_{j=1}^{N+1}\left(\left(\int_{D_k} \ell_j^k(x)\ell_i^k(x)\,dx\right)\frac{\partial u_j^k(t)}{\partial t}\right) - \sum_{j=1}^{N+1}\left(\left(\int_{D_k} \ell_j^k(x)\frac{\partial}{\partial x}\ell_i^k(x)\,dx\right)f_j^k(t)\right) \\ = -f_h^k(x_{k+\frac{1}{2}},t)\ell_i^k(x_{k+\frac{1}{2}}) + f_h^k(x_{k-\frac{1}{2}},t)\ell_i^k(x_{k-\frac{1}{2}}),\ \forall\, i = 1,\ldots,N+1.\end{aligned} \tag{2.19}$$

The beginning of this chapter mentions that RKDG schemes are highly localized, so to ensure that information on a given cell is coupled only to its local neighbors as well as to guarantee discrete conservation, the flux evaluations at the boundaries of $D_k$ can be replaced with a single-valued numerical flux $f^*_{k+\frac{1}{2}}$ at each interface $x_{k+\frac{1}{2}}$. According to Hesthaven [6], the flux guarantees "stability of the formulation by mimicking the flow of information in the underlying partial differential equation." Consequently, it is important to ensure that the numerical flux is consistent (i.e., single-valued) and monotone [6]. With these ideas in mind, equation (2.19) becomes

$$\sum_{j=1}^{N+1}\left(\left(\int_{D_k}\ell_j^k(x)\ell_i^k(x)\,dx\right)\frac{d\,u_j^k(t)}{dt}\right)-\sum_{j=1}^{N+1}\left(\left(\int_{D_k}\ell_j^k(x)\frac{d}{dx}\ell_i^k(x)\,dx\right)f_j^k(t)\right) \tag{2.20}$$
$$=-f^*_{k+\frac{1}{2}}(t)\ell_i^k(x_{k+\frac{1}{2}})+f^*_{k-\frac{1}{2}}(t)\ell_i^k(x_{k-\frac{1}{2}}),\ \forall\, i=1,\ldots,N+1.$$

Finally, writing the above equation as a vectorized scheme leads to the DG formulation [6] below,

$$\boldsymbol{M}^k\frac{d\,\boldsymbol{U}^k(t)}{dt}-\left(\boldsymbol{S}^k\right)^T\boldsymbol{F}^k(t)=-\boldsymbol{\ell}^k(x_{k+\frac{1}{2}})f^*_{k+\frac{1}{2}}(t)+\boldsymbol{\ell}^k(x_{k-\frac{1}{2}})f^*_{k-\frac{1}{2}}(t), \tag{2.21}$$

where

$$\boldsymbol{U}^k(t)=[u_1^k(t),\ldots,u_{N+1}^k(t)]^T,\ \boldsymbol{F}^k(t)=[f_1^k(t),\ldots,f_{N+1}^k(t)]^T,\ \boldsymbol{\ell}^k(x)=[\ell_1^k(x),\ldots,\ell_{N+1}^k(x)]^T,$$

and the mass matrix $\boldsymbol{M}^k$ and stiffness matrix $\boldsymbol{S}^k$ are expressed as

$$M_{ij}^k=\int_{D_k}\ell_i^k(x)\ell_j^k(x)\,dx,\quad S_{ij}^k=\int_{D_k}\ell_i^k(x)\frac{d}{dx}\ell_j^k(x)\,dx.$$

**Remark 2.2.1.** *It can be shown that the DG method is conservative, meaning that the numerical solution $u_h$ satisfies*

$$\frac{d}{dt}\int_\Omega u_h(x,t)\,dx=-f^*_{K+\frac{1}{2}}(t)+f^*_{\frac{1}{2}}(t). \tag{2.22}$$

## 2.3 Runge-Kutta Time-Stepping

As shown in the final vectorized DG formulation in equation (2.21), the spatial discretization in the previous section leads to a system of $K$ ordinary differential equations for $u_h^k(t)$. If we simplify the system to

$$\frac{d}{dt}u_h^k(t) = L_h(u_h^k, t), \quad k = 1, \ldots, K,$$

then we can write an explicit, high-order accurate Runge-Kutta method as follows [6]:

1. Set $v^{(0)} = u_n^k$, i.e., set the 0-th intermediate solution equal to the solution on $D_k$ at the $n$-th time step.

2. For $i = 1, \ldots, s$, where $s$ is the number of stages of the explicit RK method, compute the intermediate functions

$$v^{(i)} = \sum_{j=0}^{i-1} \alpha_{ij} v^{(j)} + \beta_{ij} \Delta t \, L_h(v^{(j)}, t_n + \gamma_j \Delta t).$$

3. Set the solution at the $n+1$-st time step to be the last intermediate solution, i.e., $u_{n+1}^k = v^{(s)}$.

The general Runge-Kutta method above is required to satisfy $\sum_{j=0}^{i-1} \alpha_{ij} = 1$ for consistency and $\alpha_{ij} > 0$, $\beta_{ij} > 0$ for total variation boundedness (TVB). Both of these conditions allow the method to be called "strong stability-preserving" (SSP) [6]. SSP RK methods are especially useful for handling conservation laws that generate shocks, since they do not introduce any additional oscillations during time-integration [6]. These RK methods are also simple and efficient to code; when computing the solution on each cell $D_k$, the method only needs information from $D_k$'s neighboring cells, regardless of the degree of the polynomial basis [2].

However, the timestep $\Delta t$ must be chosen carefully to ensure the stability of the method. It is standard to choose a timestep that satisfies the Courant-Friedrichs-Levy (CFL) condition, which generally requires that the timestep asymptotically behaves as $\mathcal{O}(h/N^2)$ when $N$ is increased or $h$, the mesh size, is decreased [6]. Thus, we choose an adaptive timestep according to

$$\Delta t = \text{CFL}\frac{\min_k\{h_k\}}{\lambda_{\max} N^2}, \tag{2.23}$$

where $\lambda_{\max} = \max_k |f'(u_h)|$ is the maximum wave speed, while the CFL number is typically chosen from $(0, 1)$. When working with a system of conservation laws, $f'(u_h)$ is replaced by the largest eigenvalue of the flux Jacobian.

## 2.4 Shock Capturing

In the original RKDG method, a slope limiter is applied to every cell to smooth out any discontinuities that may arise from shocks. Cockburn and Shu [2] lay out a generalized slope limiter using the minmod function $m$, defined as

$$m(a_1, a_2, a_3) = \begin{cases} s \min_{1\le n\le 3} |a_n| & \text{if } s = sign(a_1) = sign(a_2) = sign(a_3) \\ 0 & \text{otherwise} \end{cases} \tag{2.24}$$

as well as information from the current cell and its neighbors, including cell averages and interface values (more details in Chapter 4). However, the application of the limiter in smooth regions of the solution can degrade accuracy and slow down computation [13]. Therefore, troubled-cell indicator methods were introduced to locate which cells contained discontinuities, called troubled cells. The slope limiter is applied only to those troubled cells to smooth the solution there, thus avoiding spurious oscillations but not affecting the smooth parts of the solution. Over the past 20 years, many types of troubled-cell indicators have appeared, reviewed by papers such as Qiu and Shu [13] and Kochi and Ramakrishna [7]. The fourth chapter describes several troubled-cell indicators from the literature in detail.

Slope limiting in troubled cells generally takes place in two ways. First, the polynomial in the troubled cell can be replaced either with the cell average or a limited linear reconstruction

$$u_h^k(x) = \bar{u}_h^k + (x - x_k)(u_h^k)_x + \mathcal{O}(x - x_k)^2, \tag{2.25}$$

where $(u_h^k)_x$ is a quantity often replaced by the minmod function, with inputs depending on the cell averages and cell interface values [2, 6, 15]. However, if cells are incorrectly flagged as troubled cells, using this type of limiter can lead to a loss in accuracy [15].

Second, weighted essentially nonoscillatory (WENO) methods can reconstruct the polynomial at a higher degree on the current cell without the spurious oscillations. Typical WENO methods "reconstruct the values of the solutions at certain Gaussian quadrature points in the target cells, and then rebuild the solution polynomial from the original cell average and the reconstructed values ... through a numerical integration for the moments" [24]. However, this method needs to use information from a larger stencil of neighbors, so Zhong and Shu present a simpler WENO that uses information only from the target cell and its closest neighbors [24]. In any type of WENO method, the WENO weights are often computationally expensive to calculate, so it is especially important to use a troubled-cell indicator that does not flag more cells than necessary. So, both WENO and traditional slope limiter methods motivate finding a troubled-cell indicator that minimizes the number of flagged cells.

In addition to the solution-smoothing duo of troubled-cell indicators and slope limiters, nonlinear artificial viscosity methods present another avenue for shock capturing. Hyperbolic conservation laws, with their tendency to produce discontinuities, provide a perfect opportunity to add a viscosity term to the PDE that can "artificially inject diffusion to locally control the spurious oscillations," as observed by Schwander, Ray, and Hesthaven [17]. Viscosity methods are not explored in this thesis; see Yu and Hesthaven [23] or Discacciati, Hesthaven and Ray [3] for more details.

## 2.5 Implementation of DG Methods

Though we have presented all of the theoretical formulations for RKDG methods, translating them into reliable code poses another challenge altogether. The computation of the matrices, the choice of polynomial bases, the numerical flux, and the RK method are just a few places where smart choices must be made to ensure efficient and robust code. The code for this project was created by Deep Ray, based on [6], and we have modified it to include more troubled-cell indicators, including machine learning indicators.

First of all, it is important to efficiently compute the DG matrices, which can be done by transforming the integrals to the reference element, $D = [-1, 1]$ [6]. For example, we derive the computation of the mass matrix. Recall the transformation in equation (2.6) from $r \in D$ to $x \in D_k$:

$$x = \mathcal{T}_k(r) = \frac{h_k}{2}r + \frac{x_{k+\frac{1}{2}} - x_{k-\frac{1}{2}}}{2}.$$

Note that $\frac{dx}{dr} = \frac{h_k}{2}$. Then, the mass matrix becomes

$$M_{ij}^k = \int_{D_k} \ell_i^k(x)\ell_j^k(x)\,dx = \int_D \ell_i(r)\ell_j(r)\frac{h_k}{2}\,dr = \frac{h_k}{2}M_{ij}, \tag{2.26}$$

where $M$ is the mass matrix on the reference element. Therefore, we have

$$\boldsymbol{M}^k = \frac{h_k}{2}\boldsymbol{M} = \frac{h_k}{2}\int_D \boldsymbol{\ell}(r)\boldsymbol{\ell}^T(r)\,dr. \tag{2.27}$$

Using equation (2.13), and assuming the Vandermonde matrix $V$ is invertible, then we can say

$$\begin{aligned}\boldsymbol{M}^k &= \frac{h_k}{2}\int_D \left(V^T\right)^{-1}\boldsymbol{\phi}(r)\boldsymbol{\phi}^T(r)V^{-1}\,dr \\ &= \frac{h_k}{2}\left(V^T\right)^{-1}\left(\int_D \boldsymbol{\phi}(r)\boldsymbol{\phi}^T(r)\,dr\right)V^{-1}.\end{aligned} \tag{2.28}$$

Since we chose $\{\phi_i(r)\}_{i=1}^{N+1}$ to be orthonormal, then

$$\boldsymbol{M}^k = \frac{h_k}{2}\left(V^T\right)^{-1}V^{-1} = \frac{h_k}{2}\left(V\,V^T\right)^{-1}. \tag{2.29}$$

Consequently, computing the mass matrix depends on having a well-conditioned, easy-to-invert Vandermonde matrix. By choosing $\{\phi_i(x)\}$ as the orthonormal Legendre polynomial basis and $\{r_i\}$ as the Gauss-Lobatto quadrature points in $D$ [6], we ensure that $V$ is well-conditioned. The Vandermonde matrix is also essential in the conversion between the nodal and modal coefficients using equation (2.11), which is often necessary or convenient in the code. Another convenience with the orthonormal Legendre polynomials is that the cell average is proportional to the first modal coefficient, i.e., $\bar{u}_k = \frac{1}{\sqrt{2}}\hat{u}_1^k$. The stiffness matrix $\boldsymbol{S}^k$ can be similarly transformed to the reference element but does not result in a scaling constant [6].

We choose the numerical flux to be the Lax-Friedrichs numerical flux, defined in [6] as

$$f^{LF}(a,b) = \frac{1}{2}(f(a) + f(b) - C(b-a)), \tag{2.30}$$

where

$$C = \max_{\inf u_h(x) \leq s \leq \sup u_h(x)} \left| \frac{\partial f}{\partial u}(s) \right|.$$

The RK method used in our code is the strong-stability preserving (SSP) third order RK method, which is formulated as the following (recalling the format of Section 2.3), for each $k = 1, \ldots K$ [6]:

$$\begin{aligned} v^{(1)} &= u_n^k + \Delta t L_h(u_n^k, t_n), \\ v^{(2)} &= \frac{1}{4}\Big(3u_n^k + v^{(1)} + \Delta t L_h(v^{(1)}, t_n + \Delta t)\Big) \\ u_{n+1}^k = v^{(3)} &= \frac{1}{3}\Big(u_n^k + 2v^{(2)} + 2\Delta t L_h(v^{(2)}, t_n + \tfrac{1}{2}\Delta t)\Big) \end{aligned} \tag{2.31}$$

The troubled cell indicators are defined in Chapter 4. Once the troubled cells are indicated, we use the classical MUSCL reconstruction procedure for slope limiting [6]. Recall equation (2.25) for the limited linear reconstruction of $u_h^k(x)$; the MUSCL reconstruction follows the same form, but the minmod function from equation (2.24) is used to define

$$\tilde{u}_h^k(x) = \bar{u}_k + (x - x_k)\, m\left(s_k, \frac{\bar{u}_k - \bar{u}_{k-1}}{h}, \frac{\bar{u}_{k+1} - \bar{u}_k}{h}\right), \tag{2.32}$$

where $s_k$ is the slope, or derivative, at $x_k$.

# Chapter 3: Machine Learning Methods for New Troubled Cell Indicators

A large number of troubled-cell indicators have been designed over the past two decades [7,13]. However, most of these depend on problem-dependent parameters, which need to be tuned empirically. The works of Ray and Hesthaven [15,16] have shown that deep neural networks hold promise for learning an accurate, efficient troubled cell indicator function. This thesis builds on their work by evaluating other machine learning (ML) methods, like support vector machines (SVMs) and decision trees, as possible troubled cell indicators.

We consider different types of ML models trained using a supervised learning framework. Supervised learning involves generating a suitable labeled training dataset and using it minimize a loss or objective function that compares predicted and true labels. While the ML model is trained to predict well on the training dataset, it must also be able to accurately predict responses for data outside the training set. Within the RKDG context, the ML model must be able to flag the troubled cells accurately enough that applying the slope limiter in the flagged cells prevents the formation of spurious oscillations.

Our goal is to construct ML models, trained once on the training set, that reduce the overall cost of the RKDG method by flagging only the necessary cells to which slope limiter is applied. We also hope that the ML models can be applied to a variety of problems without reliance on problem-dependent parameters. On the other hand, one of the main issues with ML methods is their interpretability; often, methods like neural networks are used as a "black box" for solving problems without actually being able to track the decision process(es) being made. Ray and Hesthaven's neural network is quite large with five hidden layers [15], so it is difficult to determine the output's relationship to the inputs. Thus, we hope that using an SVM or decision tree may give some insights into the nature of troubled cell indication.

## 3.1 Support Vector Machines

Support vector machines (SVMs) aim to construct a hyperplane in a high-dimensional space to divide the data into two classes (or more, if solving a multi-classification problem). The ideal hyperplane has the largest distance to the nearest training data points of either class, called the margin. A subset of training points on or within the margin, called the support vectors, form the decision function that then classifies a data point as 1 or -1, though in our implementation we use 0 instead of -1. Fig. 3.1 visualizes these concepts.

To find this ideal hyperplane, the SVM algorithm tries to maximize the margin while penalizing samples that are misclassified or within the margin boundary. Mathematically, given training vectors $x_i \in \mathbb{R}^m$, $i = 1, \ldots n$ with labels given in the vector $y \in \{-1, 1\}^n$, the following optimization problem tries to find $w \in \mathbb{R}^m$ and $b \in \mathbb{R}$ such that the prediction $sign(w^T \phi(x) + b)$ is correct for most samples:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i, \tag{3.1}$$

subject to $y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i,$

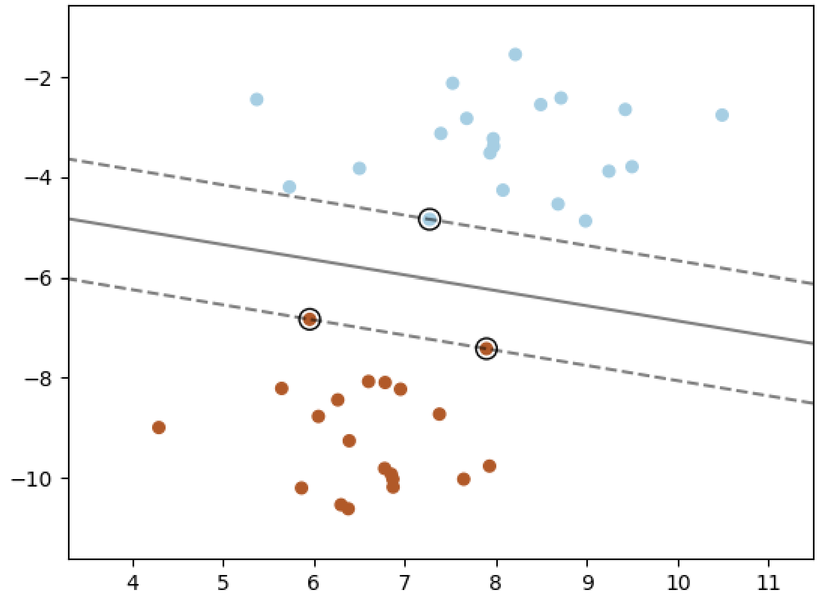



Figure 3.1: Visualization of SVM's hyperplane, margin, and support vectors. The hyperplane is the solid gray line separating the two classes of data. The margins are the dashed lines, representing the distance between the hyperplane and the closest points of either class. The support vectors are the circled data points that lie on the margins. Source: sci-kit learn documentation.

where $\zeta_i \geq 0,\ i \ldots n$ are the distances that samples are allowed to be from their correct margin boundary, since problems are not always perfectly separable. These distances are controlled by the penalty term $C$.

Note also that $\phi(x_i)$ represents a possible mapping of the support vectors to a higher dimensional space. More complex problems are likely not able to be separated by a linear hyperplane, so SVMs can use a kernel trick to transform points to a higher dimension where linear hyperplane is then formed. We define the kernel as the inner product $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$. Commonly used nonlinear kernels are listed in Table 3.1.

To find the ideal decision boundary, the SVMs typically solve the dual problem to (3.1):

$$\min_{\alpha} \frac{1}{2}\alpha^T Q\alpha - e^T\alpha, \qquad (3.2)$$
$$\text{subject to } y^T\alpha = 0,$$

where $e$ is the vector of all ones, $0 \leq \alpha_i \leq C$, $i = 1, \dots, n$ are called the dual coefficients, and $Q \in \mathbb{R}^{(n \times n)}$ is a positive semidefinite matrix defined by $Q_{ij} = y_i y_j K(x_i, x_j)$. Once this optimization problem is solved for $\alpha$, then the decision function below, which sums over the support vectors, can be used for the prediction $y$ of a given sample $x$:

$$y = \text{sign}\left(\sum_{i=1}^{N_{SV}} y_i \alpha_i K(x_i, x) + b\right). \qquad (3.3)$$

| **Kernel** | $\boldsymbol{K(x_i, x_j)}$ |
|---|---|
| Polynomial | $(\gamma(x_i^T x_j) + r)^d$ |
| Radial Basis Function (RBF) | $\exp(-\gamma\|x_i - x_j\|^2)$ |
| Sigmoid | $\tanh(\gamma(x_i^T x_j) + r)$ |

Table 3.1: Common nonlinear kernel functions for SVMs.

### 3.1.1 RBF Kernel Parameters

This thesis will use the radial basis function (RBF) kernel for the troubled-cell SVMs, since RBF kernels can capture more complicated structures of classification in complex datasets. To understand the RBF kernel and optimize its performance, it is important to understand the two parameters, $C$ (from the minimization problem) and $\gamma$ (see Table 3.1).

$C$ is the regularization parameter from equation (3.1), controlling the tradeoff of training sample misclassification and decision surface complexity. As $C$ increases, the margin decreases while the decision function attempts to classify all training samples correctly. As $C$ decreases, the margin will increase and the decision surface becomes simpler, possibly at the cost of training accuracy. Also, larger $C$ values cause longer training times, while smaller $C$ values cause a higher number of support vectors and thus longer prediction times. The other parameter, $\gamma$, is the inverse of the radius of influence of the support vectors. If $\gamma$ is small, then the support vectors have a large influence on the points around them; otherwise, when $\gamma$ is large, the support vector does not influence as many surrounding points. However, care must be taken when deciding $\gamma$ - if it is too large, the model will overfit the data, and if it is too small, the model will be underfit. Therefore, intermediate values of both $C$ and $\gamma$ are usually best, and a grid search-based cross validation strategy is recommended to find the best parameters for the model and data.

## 3.2 Decision Trees

Decision trees are another type of supervised learning method that can be used for classification. A decision tree is made up of split nodes, which make boolean decisions (true or false) to split the data based on a threshold for a certain feature of the input data, and leaf nodes, which end the tree and label the data. The thresholds and features at each nodes are chosen such that samples with the same labels are grouped together. Once the decision tree is trained, a test sample will evaluate its features depending on the threshold at each split node and decide to go left (true) or right (false), repeating the process until it reaches a leaf and is labeled.

One of the most important hyperparameters of a decision tree is its maximum depth, which is the maximum number of nodes a sample must go through before reaching a leaf node. However, very deep trees can overfit the data and may not generalize, so it is important to limit the maximum depth. Decision trees also have a regularization parameter $C$, just like SVMs. These hyperparameters will be tuned using a grid search, similar to SVMs.

Although decision trees are fast, easy to interpret, and easy to visualize, they may not learn a nonlinear classification function well since they represent a piecewise constant approximation of the data. In addition, the data set must have balanced classes to prevent creation of a biased tree. We will explain in Chapter 4 why decision trees do not work well for troubled cell classification.

# Chapter 4: A Review of Troubled Cell Indicators

There are now many existing troubled cell indicators, and several reviews have been done [7, 13] to analyze which ones work well, indicate more troubled cells, and generalize well across problems. However, many of them depend on problem-dependent parameters, so methods such as outlier detection [21] and machine learning [15] reduce reliance on these parameters. We choose a selection of troubled cell indicators, described in this chapter, that have worked best in the comparisons [7, 13]: the minmod-TVB, KXRCF, Fu and Shu, and Ray and Hesthaven indicators, as well as the outlier detection method by Vuik and Ryan that presents an alternative to conventional machine learning methods. The new SVM and decision tree indicators are then described near the end of this chapter. It should also be noted that this thesis is focused only on the troubled-cell indicator aspect, so we use the minmod-TVB slope limiter after each troubled cell indicator is applied.

## 4.1 Existing Troubled Cell Indicators

For several of the indicators used in this section, the following notations will be used. The average of the solution on cell $D_k$ is $\bar{u}_k$. The left interface value of the solution in $D_k$ (i.e., the first nodal coefficient when discretized) is $u^+_{k-\frac{1}{2}}$, and the right interface value of the solution in $D_k$

(i.e., the last nodal coefficient) is $u^-_{k+\frac{1}{2}}$. Also, recall the minmod function from equation (2.24):

$$m(a_1, a_2, a_3) = \begin{cases} s \ \min(|a_1|, |a_2|, |a_3|), & \text{if } s = sign(a_1) = sign(a_2) = sign(a_3) \\ 0 & \text{otherwise} \end{cases}.$$

1. **Minmod indicator**: This method takes advantage of the minmod function above, which sets the slope in the current cell to zero if the signs of the slopes in the three-cell stencil are different, signifying an oscillation, and otherwise returns the minimal slope [6]. To implement this method, first compute

$$\begin{aligned} \tilde{u}^+_{k-\frac{1}{2}} &= \bar{u}_k + m\Big(\bar{u}_k - u^+_{k-\frac{1}{2}}, \bar{u}_k - \bar{u}_{k-1}, \bar{u}_{k+1} - \bar{u}_k\Big), \\ \tilde{u}^-_{k+\frac{1}{2}} &= \bar{u}_k - m\Big(u^-_{k+\frac{1}{2}} - \bar{u}_k, \bar{u}_k - \bar{u}_{k-1}, \bar{u}_{k+1} - \bar{u}_k\Big). \end{aligned} \tag{4.1}$$

If the modifications in equation (4.1) above change either of $D_k$'s interface values, meaning if $u^+_{k-\frac{1}{2}} \neq \tilde{u}^+_{k-\frac{1}{2}}$ or $u^-_{k+\frac{1}{2}} \neq \tilde{u}^-_{k+\frac{1}{2}}$, then $D_k$ is marked as a troubled cell [15]. When paired with a slope limiter, this method ensures the solution is total variation diminishing in the mean (TVDM) [6]. However, this indicator also flags cells that contain a smooth extrema, leading to increased computational cost and limited local accuracy [15]. Therefore, we use only the minmod-TVB indicator described in the next paragraph in our comparison.

2. **The minmod-TVB indicator**: Cockburn and Shu [1,18] modified the minmod indicator and slope limiter to require the solution to be only total variation bounded (TVB), a less strict condition than TVDM. The structure is mostly the same as the minmod indicator, but the minmod

function is replaced with $m_{tvb}$, defined as

$$m_{tvb}(a_1, a_2, a_3; h, M) = \begin{cases} a_1 & \text{if } |a_1| \le Mh^2, \\ m(a_1, a_2, a_3) & \text{otherwise,} \end{cases} \tag{4.2}$$

where $m$ is the minmod function and $M > 0$ is a constant. Thus, the indication variables from equation (4.1) are replaced by

$$\begin{aligned} \tilde{u}^{+}_{k-\frac{1}{2}} &= \bar{u}_k - m_{tvb}\Big(\bar{u}_k - u^{+}_{k-\frac{1}{2}}, \bar{u}_k - \bar{u}_{k-1}, \bar{u}_{k+1} - \bar{u}_k\Big), \\ \tilde{u}^{-}_{k+\frac{1}{2}} &= \bar{u}_k + m_{tvb}\Big(u^{-}_{k+\frac{1}{2}} - \bar{u}_k, \bar{u}_k - \bar{u}_{k-1}, \bar{u}_{k+1} - \bar{u}_k\Big), \end{aligned} \tag{4.3}$$

and $D_k$ is marked as a troubled cell if either $u^{+}_{k-\frac{1}{2}} \neq \tilde{u}^{+}_{k-\frac{1}{2}}$ or $u^{-}_{k+\frac{1}{2}} \neq \tilde{u}^{-}_{k+\frac{1}{2}}$.

Though the minmod-TVB indicator solves the minmod indicator's problem with smooth extrema, it also introduces the problem-dependent parameter $M$. $M$ can be estimated for scalar problems - it is proportional to the curvature of the initial condition at smooth extrema [13] - but $M$ is more difficult to estimate for systems. If $M$ is too small, the indicator will mark more cells than necessary as troubled cells, increasing computational cost; if $M$ is too big, the solution may generate spurious oscillations [13].

3. **KXRCF indicator**: This shock-detection technique by Krivodonova et al. [8] is based on a strong superconvergence at the outflow boundary of each cell in smooth regions of the DG method; detecting discontinuities in density or entropy prohibits the application of limiting in smooth regions of the solution. For the troubled cell indicator, first partition the boundary of a cell $D_k$ into two portions using the normal vector $\vec{n}$ and the flow $\vec{v}$ : $\delta D_k^-$, where the flow is into

$D_k$ ($\vec{v} \cdot \vec{n} < 0$); and $\delta D_k^+$, where the flow is out of $D_k$ ($\vec{v} \cdot \vec{n} < 0$). The discontinuity detector is defined as [8]:

$$\mathcal{I}_k = \frac{\left| \int_{\delta D_k^-} u_h^k - u_h^{n_k} \, ds \right|}{h^{(N+1)/2} \, |\delta D_k^-| \, \|u_h^k\|}, \tag{4.4}$$

where $u_h^{n_k}$ is the solution on cell $D_{n_k}$, $D_{n_k}$ is the neighbor of $D_k$ on the side of $\delta D_k^-$, $h$ is the radius of the circumscribed circle in cell $D_k$, $N$ is the degree of the solution polynomial, and the norm in one-dimension is based on an cell average [13]. If $\mathcal{I}_k > M$, where $M$ is some problem-dependent parameter, then cell $D_k$ is marked as a troubled cell.

4. **Fu and Shu indicator**: In 2017, Fu and Shu proposed a troubled cell indicator that depends on both the regular cell averages and the extended cell averages of the neighboring solutions into the target cell [4]. Fu and Shu's paper describes the formulation for two dimensions, so we translate it to one dimension for use in this thesis. Let us define $D_k$ as the target cell, with $D_{k-1}$ and $D_{k+1}$ making up the neighboring-cell stencil. Then, denote the DG polynomial solutions on these cells as $p_{k-1}$, $p_k$, and $p_{k+1}$. For each target cell, consider the quantity

$$I_{D_k} = \frac{|\bar{\bar{p}}_k - \bar{\bar{p}}_{k-1}| + |\bar{\bar{p}}_k - \bar{\bar{p}}_{k+1}|}{\max\{|\bar{p}_{k-1}|, |\bar{p}_k|, |\bar{p}_{k+1}|\}}, \tag{4.5}$$

where

$$\bar{\bar{p}}_i = \frac{1}{h_k} \int_{D_k} p_i(x) \, dx, \quad \bar{p}_i = \frac{1}{h_i} \int_{D_i} p_i(x) \, dx, \quad \text{for } i = k-1, k, k+1.$$

In other words, $\bar{\bar{p}}$ is the cell average of the function $p$ on the target cell (extended from the neighboring cell if applicable), while $\bar{p}$ is the cell average of the function $p$ on its own corresponding cell [4]. Note that $\bar{\bar{p}}_k = \bar{p}_k$. If $I_{D_k} > C_N$, where $C_N$ depends on the polynomial degree $N$, then $D_k$ is marked as a troubled cell. The constants $C_N$ used by Fu and Shu are reported in Table 4.1.

| $N$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $C_N$ | 0.05 | 0.1 | 0.25 | 0.5 | 2.5 | 5 |

Table 4.1: Values of $C_N$ for each degree $N$ as described in [4].

5. **Vuik and Ryan outlier detection**: In their 2016 paper, Vuik and Ryan developed a parameter-free indicator based on Tukey's boxplot approach. They argued that troubled-cell indication reduces to detecting outliers in vectors of certain indicator values, since cells with discontinuities typically correspond to the locations where the indicator value is suddenly much different than the neighboring values [21]. Thus, the following troubled-cell indicator algorithm based on outlier detection was developed:

1. Send in an appropriate troubled-cell indication vector, $D$, which has $K = 2^n$ elements.

2. Split $D$ into local vectors $d$, each of length $2^p$ for $p \in \{2, \dots, n\}$ (Vuik and Ryan suggest $p = 4$ to capture the right amount of information in a boxplot [21]).

3. Sort each local vector $d$ to obtain $d^s = (d_1^s, \dots, d_{2^p}^s)$.

4. Compute the first and third quartiles:

$$Q_1 = \frac{d^s_{2^p/4} + d^s_{2^p/4+1}}{2}, \quad Q_3 = \frac{d^s_{3\cdot 2^p/4} + d^s_{3\cdot 2^p/4+1}}{2}.$$

5. The outer fences of a vector are $[Q_1 - 3(Q_3 - Q_1), Q_3 + 3(Q_3 - Q_1)]$, proposed by Tukey and chosen such that only extreme outliers (0.0002% of normally distributed data) lie outside the outer fences. Therefore, detect $d_j^s$ in the lowest 25% of $d^s$ if $d_j^s < Q_1 - 3(Q_3 - Q_1)$, and $d_j^s$ in the last 25% of $d^s$ if $d_j^s > Q_3 + 3(Q_3 - Q_1)$.

6. Finally, to keep the importance of spatial information, ignore the detected outliers in the left half of the local vector (spatially, not sorted) when they are not detected using the left-neighboring vector's outer fences, and similarly test the detected coefficients in the right half of the local vector.

Vuik and Ryan applied this algorithm to indication variables from several indicator methods, but we focus on their application to the minmod-TVB indicator.

For the **Vuik and Ryan TVB** indicator, two quantities were defined for each cell $D_k$:

$$\tilde{u}_k = u^-_{k+\frac{1}{2}} - \bar{u}, \text{ and } \tilde{\tilde{u}}_k = \bar{u} - u^+_{k-\frac{1}{2}}. \tag{4.6}$$

Since DG coefficients within cell $D_k$ tend to differ substantially from their neighbors if $D_k$ contains a discontinuity, then Vuik and Ryan take two indicator variable vectors on which their outlier detection algorithm is to be performed: $D_1 = (\tilde{u}_1, \ldots, \tilde{u}_{2^n})^T$ and $D_2 = (\tilde{\tilde{u}}_1, \ldots, \tilde{\tilde{u}}_{2^n})^T$. Cell $D_k$ is marked as a troubled cell if either $\tilde{u}_k$ or $\tilde{\tilde{u}}_k$ are detected as outliers [21].

6. **Neural Network indicator**: Ray and Hesthaven [15] created a deep neural network, following the multi-layer perceptron (MLP) framework, that functions as a troubled cell indicator. The input to their network is the vector $(\bar{u}_{k-1}, \bar{u}_k, \bar{u}_{k+1}, u^+_{k-\frac{1}{2}}, u^-_{k+\frac{1}{2}})^T$, the same values as the inputs to the minmod and minmod-TVB indicators. Ray and Hesthaven's 2018 network has 5 hidden layers with widths varying from 256 to 128, then to 64, then 32, and finally 16 before moving to the output layer. The leaky ReLU (Rectified Linear Unit) activation was used throughout [15]. The output layer produces $\boldsymbol{X}_O = (X_1, X_2)^T$, which is then fed into the softmax output

function to get the probabilities that cell $D_k$ is a troubled cell or not. Specifically,

$$(\hat{Y}_1, \hat{Y}_2) = \left( \frac{e^{X_1}}{e^{X_1} + e^{X_2}}, \frac{e^{X_2}}{e^{X_1} + e^{X_2}} \right), \tag{4.7}$$

where $\hat{Y}_1$ is the probability that $D_k$ is a troubled cell. If $\hat{Y}_1 > \hat{Y}_2$, then $D_k$ is flagged.

The cross entropy cost function, given by

$$C = -\frac{1}{S} \sum_{k=1}^{S} Y_1^k \ln(\hat{Y}_1^k) + Y_2^k \ln(\hat{Y}_2^k), \tag{4.8}$$

where $S$ is the number of samples used for training and $(Y_1^k, Y_2^k)^T$ is the true troubled-cell probability distribution for sample $k$, is used to train the network. To avoid overfitting the training data, the cost function is regularized to penalize the network weights $\boldsymbol{W}$, leading to the modified minimization of

$$\tilde{C} = C + \beta \|W\|_2^2, \quad \beta \geq 0. \tag{4.9}$$

Ray and Hesthaven used a stochastic optimization algorithm that utilizes mini-batches in order to gain faster convergence. The training set in this thesis is similar to the training set used for the neural network.

We consider two versions of Ray and Hesthaven's neural networks: the first, labeled **MLP_v1** in Chapter 5, is from their 2018 paper; the second, labeled **MLP_v2** in Chapter 5, contains only three hidden layers instead of five and was obtained by Ray and Meng in 2024 by replicating the behavior of MLP_v1 using the smaller network, i.e., via network compression.

## 4.2 New Troubled Cell Indicators

### 4.2.1 ML Method Implementation

While Ray and Hesthaven's neural networks used cell averages and polynomial interface differences as inputs for their neural network, here we decide to evaluate the performance of a ML model that receives the nodal coefficients as input. The nodal coefficients are already computed before troubled-cell indication, so no additional calculations are necessary before plugging into the ML method.

#### 4.2.1.1 Building the Training Set

To build the training set, we must create data that resembles the nodal coefficients originating from the DG formulation for both smooth and discontinuous regions. We choose the input to be a vector $\boldsymbol{x} \in \mathbb{R}^{3(N+1)}$ containing the $N+1$ nodal coefficients per cell of the 3-cell stencil comprising the target cell and its two immediate neighbors. For each input vector, the output label will be 1 if the cell is a troubled cell, or 0 otherwise. We generate a library of well-defined functions, both smooth and discontinuous, from which the input vectors of nodal coefficients will be extracted. This library is based on the functions used to train the network in [14] and is summarized in Table 4.2.

First, we use two types of cubic equations so that the model has a "knowledge" of smooth, polynomial-like functions. Second, we include a sine function with varying frequency up to 10; the model should be able to capture high-frequency smooth waves without labeling them as discontinuities. Third, and most importantly, we include a class of discontinuous, piecewise linear functions with different slopes. We randomize the location of the discontinuity within the domain so the ML model learns how to detect troubled cells with different discontinuity locations. All of these functions are defined on the interval $[-1,1]$.

| No. | Type | $u(x)$ | Parameters |
|---|---|---|---|
| 1 | Smooth | $ax^3+bx^2+cx+d$ | $a,b,c,d \in \mathbb{U}[-10,10]$ |
| 2 | Smooth | $(x-a)(x-b)(x-c)+d$ | $a,b,c,d \in \mathbb{U}[-2,2]$ |
| 3 | Smooth | $\sin(2\pi(ax)+b)$ | $a \in \mathbb{U}[-10,10],\ b \in \mathbb{U}[-2,2]$ |
| 4 | Discontinuous | $\begin{cases} ax+b & \text{if } x \leq n \\ cx+d & \text{if } x > n \end{cases}$ | $a,b,c,d \in \mathbb{U}[-5,5],\ n \in \mathbb{U}[-1,1]$ |

Table 4.2: Library of functions used to create the training data.

For each function in the library, the process for creating the dataset is generally as follows:

1. Pick a function $u(x)$ from the library of functions on the domain $[-1,1]$.

2. Pick a number $K$ and discretize $[-1,1]$ into $K$ cells. If $u(x)$ is a sine function, pick $K$ large enough to prevent aliasing of high-frequency waves.

3. Project $u(x)$ to $u_h(x)$, a discontinuous polynomial of degree $N$ on each cell (see section 4.2.1.2), using the Vandermonde matrix to get the nodal coefficients of $u_h(x)$.

4. For each $k=2,\ldots,K-1$, create an input vector of length $3(N+1)$ of nodal coefficients from cells $k-1,k,k+1$.

5. Scale each input vector to have maximum 1 according to the following function [16]:

$$Scale(x) = \begin{cases} \frac{x}{\max_i |x_i|} & \text{if } \max_i |x_i| > 1 \\ x & \text{otherwise} \end{cases}$$

6. Create the output labels, either 1 or 0, for each input vector.

This process is repeated for as many smooth and discontinuous functions as we want to put in the dataset. Additional details about the composition of the dataset are included in Section 4.2.1.3.

**Remark 4.2.1.** *We create separate datasets for degrees $N = 1, 2, 3$ and train separate models on them, because the size of the nodal coefficient input vector changes depending on $N$.*

For smooth functions, the output labels should all be zeros. However, the process for discontinuous functions is more involved. To be conservative, we set the output label equal to one if the discontinuity at $x = n$ (from Table 4.2) occurs within an $\varepsilon$ range of the center cell's boundaries. If the discontinuity is farther away from the center cell, then the center cell is not marked as a troubled cell.

**Remark 4.2.2.** *At first, we trained the SVM with $\varepsilon = 1/6$; but, after testing, we determined that $\varepsilon = 1/2$ is more robust at capturing troubled cells. Since the neural network was also trained using $\varepsilon = 1/2$, we can fairly compare the SVM's performance with that of the network.*

In addition, since the constants are randomly generated, it is possible that the two piecewise linear functions are very close together at the discontinuity, making the function appear almost continuous. So, if the difference between $ax + b$ and $cx + d$ at the discontinuity $x = n$ is less than one percent of the sum of the absolute variation of each piecewise function between the discontinuity and the boundary (i.e., $|a(n - (-1))| + |c(n - 1)|$), then we make the output label 0. These choices for the discontinuous library functions enable the ML method to perform well with troubled cell indication.

#### 4.2.1.2 Projecting $u(x)$ to $u_h(x)$

To find the coefficients of the projection from $u(x)$ to $u_h(x) \in V_h^N$ (recall equation (2.3)), we find the best approximation $u_h(x)$ for $u(x)$ using the $L_2$ norm. On the k-th cell $D_k$, we want $u_h^k(x)$ such that

$$u(x) \approx u_h^k(x) = \sum_{i=1}^{N+1} c_i \phi_i(x) = \min_{\tilde{u} \in \text{ span}\phi_i} ||u - \tilde{u}||_2^2, \tag{4.10}$$

where the $\phi_i$ is a polynomial in the orthonormal Legendre basis, and $c_i$ is the i-th modal coefficient for $u_h^k(x)$. If we define

$$\mathcal{L}(c) = ||u - \tilde{u}||_2^2 = \int_{D_k} (u(x) - \sum_{i=1}^{N+1} c_i \phi_i)^2 \, dx, \tag{4.11}$$

then solving for the projection coefficients $c_i$ becomes a minimization problem of $\mathcal{L}(c)$. Therefore, we take the derivative of $\mathcal{L}(c)$ with respect to each coefficient and set it equal to 0, resulting in

$$\frac{\partial \mathcal{L}(c)}{\partial c_j} = -2 \int_{D_k} \Big( u(x) - \sum_{i=1}^{N+1} c_i \phi_i \Big) \phi_j(x) \, dx = 0, \quad \forall j = 1, \ldots, N+1. \tag{4.12}$$

Manipulating the above equation leads to

$$\sum_{i=1}^{N+1} c_i \int_{D_k} \phi_i(x) \phi_j(x) \, dx = \int_{D_k} u(x) \phi_j(x) \, dx. \tag{4.13}$$

If we let $M_{ij}^k = \int_{D_k} \phi_i(x)\phi_j(x) \, dx$ be the mass matrix, $b_j = \int_{D_k} u(x)\phi_j(x) \, dx$ be the right hand side of the above equation, and $\boldsymbol{c} = [c_1, \ldots, c_{N+1}]^T$, then we can rewrite the previous equation as

$$M^k \boldsymbol{c} = \boldsymbol{b}.$$

Note that $M$, the mass matrix on the reference element $D = [-1, 1]$, is then the identity matrix due to orthonormality - this nice property is the reason for finding the modal coefficients first. Therefore, since $M^k = \frac{h}{2}M = \frac{h}{2}I$ by integrating over the reference element, the previous equation is reduced to the minimization solution,

$$\boldsymbol{c} = \frac{2}{h}\boldsymbol{b}. \tag{4.14}$$

To find $\boldsymbol{b}$, we transfer the integral for $b_j$ to the reference element $[-1, 1]$ to get

$$b_j = \int_D u(r)\phi_j(r) \cdot \frac{h}{2}\, dr. \tag{4.15}$$

If we discretize $u(r)$ as $u(r_i) = u_i$, where $\{r_i\}_{i=1}^{N+1}$ are the Gauss-Lobatto quadrature nodes, and if we find the Gauss-Lobatto quadrature weights $w_i$ [6], then we change the previous integral to the following sum:

$$b_j = \frac{h}{2}\sum_{i=1}^{N+1} u_i\, \phi_j(r_i)\, w_i. \tag{4.16}$$

Recall the Vandermonde matrix $V_{ij} = \phi_j(r_i)$; so, we can vectorize this sum to compute $\boldsymbol{b}$ as

$$\boldsymbol{b} = \frac{h}{2}V^T(\boldsymbol{u} \odot \boldsymbol{w}), \tag{4.17}$$

where $\odot$ represents component-wise multiplication. Therefore, when plugging $\boldsymbol{b}$ into the equation for $\boldsymbol{c}$, the constants cancel out, and we are left with the relatively easy computation for the modal coefficients,

$$\boldsymbol{c} = V^T(\boldsymbol{u} \odot \boldsymbol{w}). \tag{4.18}$$

After these modal projection coefficients are computed, the Vandermonde matrix and equation (2.11) are used to find the nodal coefficients for $u(x)$ on the k-th cell.

#### 4.2.1.3 SVM Training

After the structure of the training dataset outlined in Section 4.2.1.1 was built, the SVM was created using the Sci-kit Learn ML library in Python. The input data and label data were imported from MATLAB to Python, where they were split 80-20 into training and test sets, respectively. A grid search of hyperparameters was implemented on the training set to find the best combination of $C$ and $\gamma$ values for the model, using a exponentially spaced range of $C$ values from $10^0$ to $10^5$ and $\gamma$ values from $10^{-4}$ to $10^1$. During the grid search, K-fold cross validation was used; the shuffled training set was split randomly into 5 folds, where each parameter combination was trained on some majority of the fold. Then, the trained model was tested for cross-validation accuracy on the left-out part of the fold, and the cross-validation accuracies from each fold were then averaged. The model with the parameter combination that had the highest mean cross-validation accuracy was then trained on the entire training set and tested on the test set.

After running this process with different variations in the number of smooth and discontinuous functions, it was determined that a more unbalanced dataset, meaning the inclusion of only a small fraction of 1 labels (about 1/8 of the whole dataset), produced better cross-validation accuracy and test accuracy. The final numbers for the training set are detailed in Table 4.3, with rows corresponding to the functions in Table 4.2. $M$ is the number of individual functions of each type, and $K$ is the number of cells in the discretization of those functions. Since the input samples are nodal coefficients for a three-cell stencil, there will only be $K-2$ samples per individual function. In addition, note that the mesh is more refined for the sine functions to prevent aliasing of high-frequency oscillations. The total number of input vectors is 11,830.

| **Type** | **M** | **K** | **Total Samples ($M * (K-2)$)** |
|---|---|---|---|
| Cubic 1 | 34 | 100 | 3332 |
| Cubic 2 | 34 | 100 | 3332 |
| Sine | 17 | 200 | 3366 |
| Discontinuous | 1800 | 3 | 1800 |

Table 4.3: Number of samples per type of function in the data set library.

Table 4.4 shows the results of the grid search and subsequent training for each SVM (degrees $N = 1, 2, 3$) with the 1/8 unbalanced dataset. After training, the SVM model was exported from Python to Matlab.

| **N** | **Best $C$** | **Best $\gamma$** | **Mean CV Accuracy** | **Number of Support Vectors** | **Test Accuracy** |
|---|---|---|---|---|---|
| 1 | 1 | 10 | 0.964180 | 1445 | 0.964497 |
| 2 | 1 | 10 | 0.968829 | 1611 | 0.964920 |
| 3 | $10^4$ | 0.1 | 0.986052 | 304 | 0.985630 |

Table 4.4: Results from training SVM for degrees $N = 1, 2, 3$.

### 4.2.2 SVM Troubled Cell Indicator

After the SVMs for degrees $N = 1, 2, 3$ were trained in Python using the Sci-kit Learn library, the SVM model was imported to the main RKDG solver code in MATLAB. For each run of the solver, the dual coefficients $y_i\alpha_i$ (specific to Sci-kit Learn's SVM implementation), support vectors $x_i$, intercept $b$, and hyperparameter $\gamma$ from the specified SVM model are saved. Then, to perform troubled cell indication, the SVM indicator receives the current nodal coefficients, formats them into input vectors $x$ of 3-cell stencils, and inputs the vectors into the RBF decision function (modified from equation (3.3)) to get the decision values $d$, given by

$$d = \sum_{i=1}^{N_{SV}} y_i \alpha_i \, e^{-\gamma \|x - x_i\|^2} + b. \tag{4.19}$$

If $d > 0$, then the cell in the center of the stencil $x$ is marked as a troubled cell.

It is worth noting that the grid search for hyperparameters and the training of the SVM need only happen once, and the hyperparameters are not tuned again. In other words, similar to the neural network, the proposed SVM approach is free from problem-dependent parameters. The same SVM is used as a troubled cell indicator across different initial conditions for different conservation laws in Chapter 5.

### 4.2.3 Decision Tree Indicator

Since decision trees require a balanced data set, a new training data set was constructed with approximately half of labels as ones and zeros for the rest. A grid search was conducted over impurity function choices (either Gini or entropy), maximum depth values from 10 to 100, and varying minimum values of samples required for split and leaf nodes from 1 to 10. However, the model with the best parameters only achieved an average cross-validation accuracy of 87.86% and a test accuracy of 88.51%; these accuracy scores are much worse than the SVM's results. By plugging the decision tree into the MATLAB solver, we confirmed that the decision tree's performance is worse than the SVM. First, the decision tree indicated many troubled cells for linear advection of a sine wave, which should have no troubled cells at all. Second, the decision tree did not indicate enough troubled cells to prevent spurious oscillations that developed around shock waves in the Burgers' equation tests. Therefore, we decide not to include the decision tree as a troubled cell indicator.

In an attempt to still use information from the decision tree, we evaluated the tree's feature importances, which rank the features that are used most in the tree's split node decisions. We hypothesized that the features with lowest importance may not be necessary in the SVM; thus, we retrained the SVM on highest-ranking two-thirds of features for $N = 1$, $2$, and $3$ and tested it as a troubled cell indicator. The feature importance SVM for $N = 1$ indicated many more troubled cells and created larger oscillations in the solution than the original SVM. For $N = 2$, the feature importance SVM still indicated more troubled cells than the original SVM, but in the $N = 3$ case, feature importance decreased the number of indicated troubled cells from the original SVM. The performance of the feature importance SVM for $N = 2$ and $N = 3$ was almost identical to the performance without feature importance, so feature importance does not provide a significant improvement to the SVM.

In addition, logistic regression was tried as an alternative to SVMs and decision trees, but the cross-validation and test accuracy scores were about the same as the decision tree. Since logistic regression and decision trees did not work well as troubled cell indicators, we conclude that the troubled cell indicator function is more nonlinear than these methods can capture.

# Chapter 5: Numerical Results

In this chapter, we demonstrate the results of using the SVM as a troubled-cell indicator for RKDG methods in several test cases, from linear advection to Burgers' equation and the Euler equations. We compare the performance of the SVM with that of the minmod-TVB, KXRCF, Fu and Shu, and Vuik and Ryan TVB indicators, as well as two of Ray and Hesthaven's neural networks. To form this comparison, we show both the numerical solutions and the time-history plots of troubled cells for each indicator and degrees $N = 1, 2, 3$ to evaluate accuracy and computational cost - if the method shows more troubled cells, it is more costly than one with less troubled cells.

Note that all indicators will use the same mesh except possibly the Vuik and Ryan TVB indicator, which must have $K$ equal to a power of 2 (see description in Chapter 4). If the mesh is 100 cells, Vuik and Ryan TVB will use 128 cells; similarly, it will use 256 instead of 200 cells.

Table 5.1 will describe which problem-dependent parameters are used for each of the test cases (except the one considered in Section 5.1.1) for the minmod-TVB, KXRCF, and Fu and Shu indicators. For minmod-TVB and KXRCF, different parameters must be found for each different initial condition and degree of the solution, which can become a time-consuming process. We limit the search for minmod-TVB parameters to 10, 100, and 1000, as done in [15]. To determine the best parameters, we choose the ones that flag the least troubled cells while giving an accurate solution free of large spurious oscillations. We reiterate that this approach to fine tuning the parameters is tedious, which is the primary motivation behind the ML-based troubled cell indicators. Note that for KXRCF, the parameters chosen represent a small range of values in which the solutions are viable, and we acknowledge that there are multiple possible good parameter values. For Fu and Shu, we use the parameters listed in Table 4.1.

| Problem | minmod-TVB M | | | KXRCF M | | | Fu and Shu C | | |
|---|---|---|---|---|---|---|---|---|---|
| | N=1 | N=2 | N=3 | N=1 | N=2 | N=3 | N=1 | N=2 | N=3 |
| Linear Adv. Multi-Wave | 100 | 1000 | 1000 | 5 | 5 | 40 | 0.05 | 0.1 | 0.25 |
| Burgers' Compound | 100 | 100 | 100 | 1 | 10 | 35 | 0.05 | 0.1 | 0.25 |
| Euler Sod | 10 | 10 | 10 | 0.25 | 2 | 5 | 0.05 | 0.1 | 0.25 |
| Euler Lax | 10 | 10 | 10 | 0.25 | 2.5 | 5 | 0.05 | 0.1 | 0.25 |
| Euler Double Raref. | N/A | 100 | 100 | N/A | 2.5 | 20 | 0.05 | 0.1 | 0.25 |

Table 5.1: Problem-dependent parameters for every test case, for indicators minmod-TVB, KXRCF, and Fu and Shu.

## 5.1 Linear Advection

First, we evaluate the numerical solutions of the linear scalar advection equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \tag{5.1}$$

where the speed $c = 1$ for all tests. The timestep is obtained by setting the CFL number to 0.2.

### 5.1.1 Sine Wave

This first test shows the advection of a smooth sine wave; there should be no discontinuities and thus no troubled cells indicated. The initial condition is $u_0(x) = \sin(10\pi x)$ for $x \in [0, 1]$, with periodic boundary conditions. We simulate until time $T = 1$, when the wave returns to its original position.

First, we investigate the robustness of the problem-dependent parameters of the minmod-TVB, KXRCF, and Fu and Shu indicators. We simulate the sine wave advection on different sized meshes of $K = 50, 100, 200$ and list the parameters that correctly did not produce any troubled cells in the following table. In general, we look for the approximately minimum parameter such that the indicator does not flag any troubled cells for this problem.

| Mesh Size | minmod-TVB M | | | KXRCF M | | | Fu and Shu C | | |
|---|---|---|---|---|---|---|---|---|---|
| | N=1 | N=2 | N=3 | N=1 | N=2 | N=3 | N=1 | N=2 | N=3 |
| $K = 50$ | 1000 | 1000 | 1000 | 5 | 18 | 0.62 | 0.4 | 0.15 | 0.02 |
| $K = 100$ | 1000 | 1000 | 1000 | 2.2 | 10 | 0.2 | 0.1 | 0.04 | 0.002 |
| $K = 200$ | 1000 | 1000 | 1000 | 1.1 | 9 | 0.04 | 0.03 | 0.01 | 0.001 |

Table 5.2: Table of approximate minimum parameters for minmod-TVB, KXRCF, and Fu and Shu that prevent indication of troubled cells for the sine wave advection problem.

It is clear from Table 5.2 that even within the same conservation law with the same initial conditions, the mesh size and degree of the solution can significantly change the problem-dependent parameter values necessary to ensure no flagged troubled cells for the KXRCF and Fu and Shu indicators. The minmod-TVB indicator is more robust to mesh size and degree changes for this problem because of the constant curvature at smooth extrema of the solution (recall Section 4.1). Table 5.2 again illustrates the problems related to indicators with problem-dependent parameters.

Searching for the right parameters involves more time and solver runs than necessary, especially since we can instead use a widely applicable machine learning method that produces virtually identical results.

**Remark 5.1.1.** *Note that for the Fu and Shu parameters, in the cases of* $K = 50$ *and* $N = 1$*,* $K = 50$ *and* $N = 2$*, and* $K = 100$ *and* $N = 1$*, the Table 5.1 parameters flag troubled cells. So, we increase them to the parameters in Table 5.2, which do not flag troubled cells. In every other case in Table 5.2, the parameters from Table 5.1 do not flag troubled cells. Thus, we are then able to decrease the parameter to find the boundary between flagged and non-flagged cells.*

After testing the indicators with problem-dependent parameters, we also examine the robustness of the SVM, neural networks, and Vuik and Ryan TVB indicators to mesh size and degree changes. In Table 5.3, we test mesh sizes of 64, 128, and 256 (since the Vuik and Ryan TVB indicator requires the mesh size to be a power of 2).

Table 5.3 shows that the SVM, neural networks, and Vuik and Ryan TVB indicators are quite good at not flagging troubled cells even as the mesh size and degree changes. However, for the $K = 64$ and $N = 1$ case, the SVM indicates more troubled cells than the neural network, while Vuik and Ryan TVB flags no troubled cells at all. In other words, on a coarse grid, with $N = 1$ and a highly oscillatory solution, it is not easy to avoid flagging cells. But with more mesh refinement or increased $N$, all of these methods perform well.

| Mesh Size | $N$ | Number of Troubled Cells per Timestep | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | SVM | | MLP_v1 | | MLP_v2 | | Vuik and Ryan TVB | |
| | | Avg | Max | Avg | Max | Avg | Max | Avg | Max |
| $K = 64$ | 1 | 30.12 | 44 | 9.39 | 36 | 8.59 | 42 | 0 | 0 |
| | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $K = 128$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $K = 256$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 5.3: Table of troubled cell average and max counts for each timestep for the SVM, both networks, and Vuik and Ryan TVB, over different mesh sizes and solution degree values.

### 5.1.2 Multi-Wave

This test case has an initial condition consisting of waves of different shapes, from linear to a stair-step wave to a quadratic. The initial condition is given by

$$u_0(x) = \begin{cases} 10(x-0.2)+1 & \text{if } 0.2 \leq x < 0.3, \\ 10(0.4-x)+1 & \text{if } 0.3 \leq x < 0.4, \\ 2 & \text{if } 0.6 \leq x < 0.8, \\ 100(x-1)(1.2-x)+1 & \text{if } 1 \leq x < 1.2, \\ 1 & \text{otherwise} \end{cases} \tag{5.2}$$

on the domain $[0, 1.4]$ with periodic boundary conditions. The solution is integrated on a mesh of $K = 100$ cells until $T = 1.4$, when the wave returns to its starting point.

**Remark 5.1.2.** *We add 1 to each of the pieces of the initial condition here because our implementation of the Fu and Shu indicator led to a divide-by-zero error when the initial condition had constant portions at 0.*

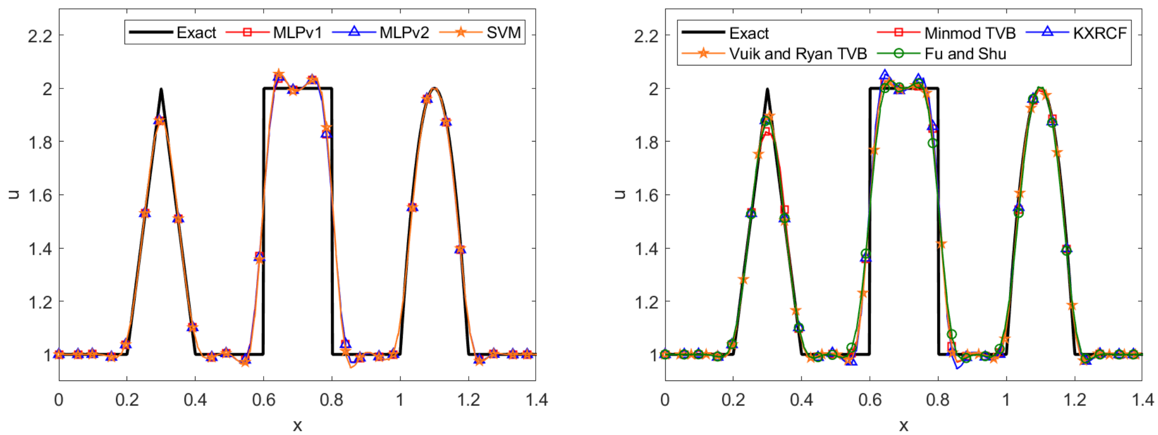
As seen in the solutions shown in Fig. 5.1, the SVM's performance is practically identical as the neural networks for $N = 1$, slightly worse for $N = 2$ due to a larger oscillation in the middle wave, and slightly better at capturing the discontinuities for $N = 3$. The SVM also performs on par with or better than the non-machine learning indicators for each degree of solution. As for the troubled cell plots in Figures 5.2 and 5.3, the SVM indicates less troubled cells than Vuik and Ryan TVB and about the same amount as Fu and Shu for $N = 1$ and $N = 2$. Interestingly, the SVM flags many more troubled cells for $N = 3$ than the other indicators. We will discuss possible issues with the $N = 3$ SVM later in this chapter.

## 5.2 Burgers' Equation

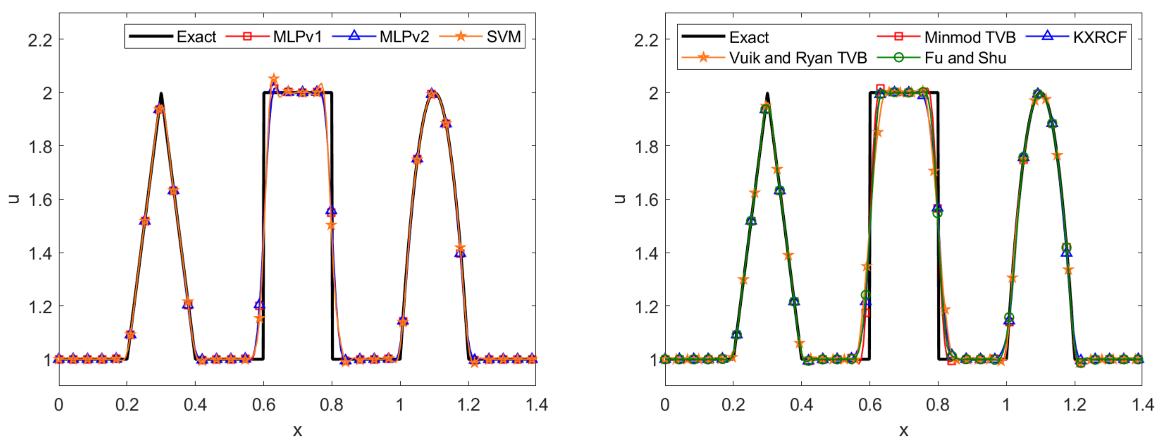
Next, we investigate performance on the nonlinear Burgers' equation:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) = 0. \tag{5.3}$$

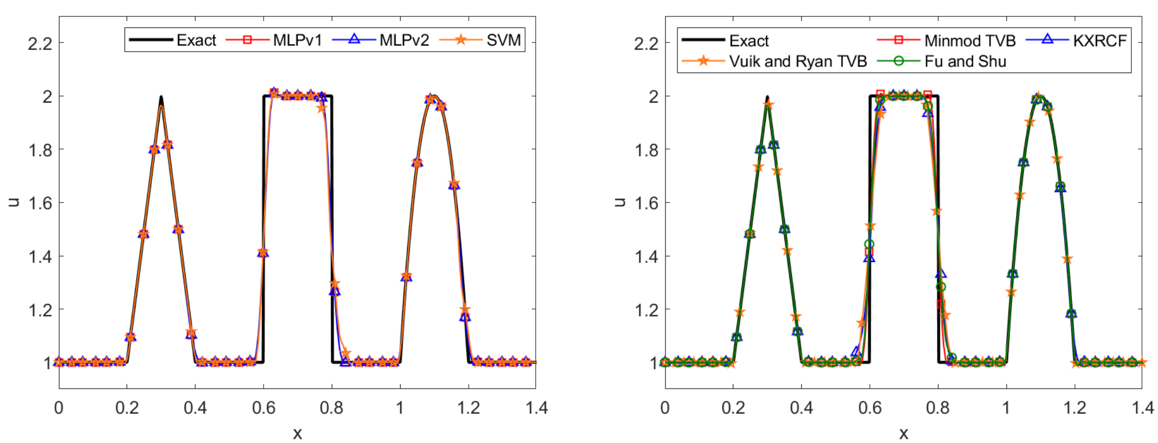
The time step is again obtained using CFL=0.2.




(a) N=1

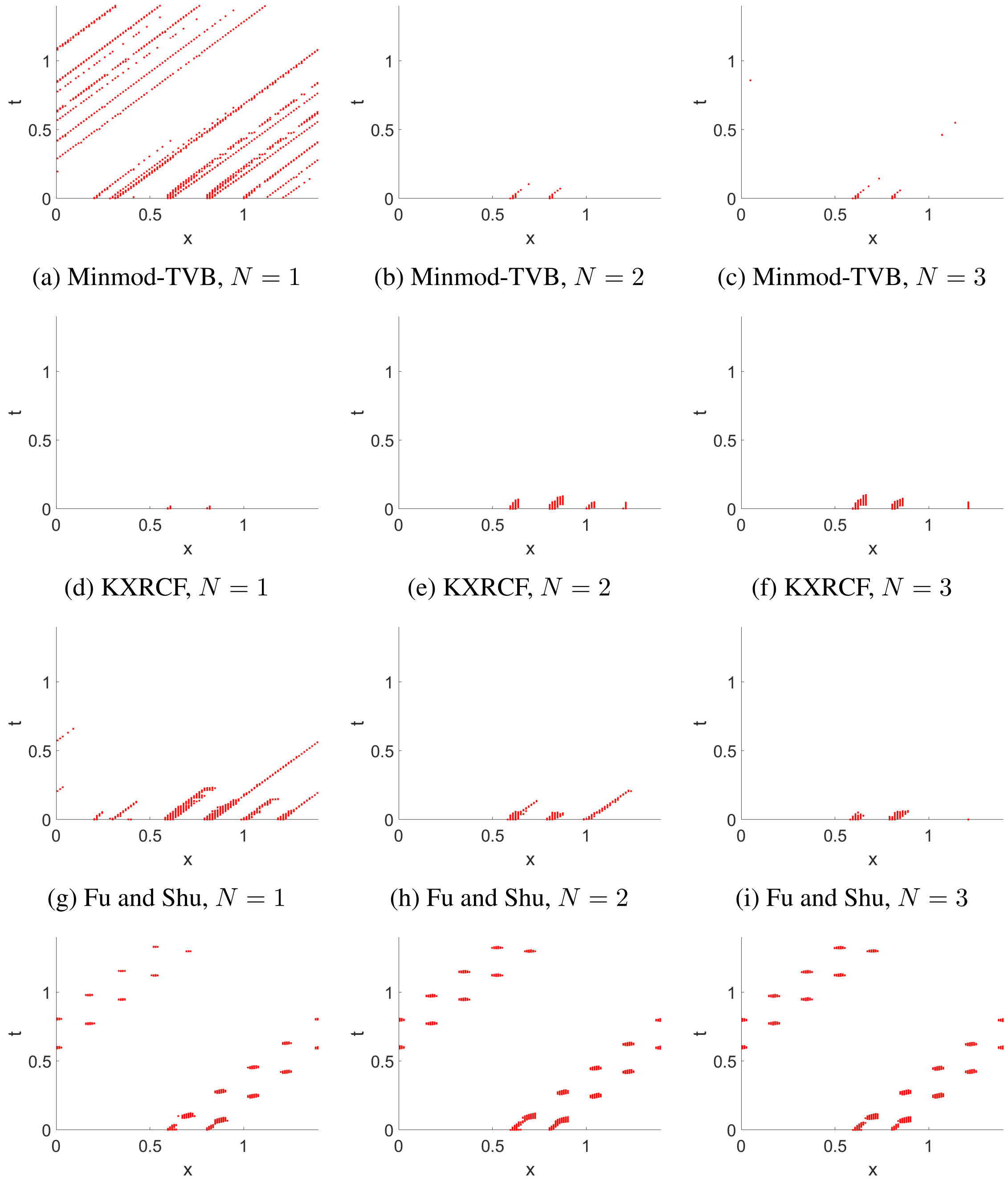



(b) N=2




(c) N=3

Figure 5.1: Solution of the multi-wave problem for linear advection, computed at $T = 1.4$ with $K = 100$. (a) degree $N = 1$ polynomial solution, (b) degree $N = 2$ polynomial solution, (c) degree $N = 3$ polynomial solution.



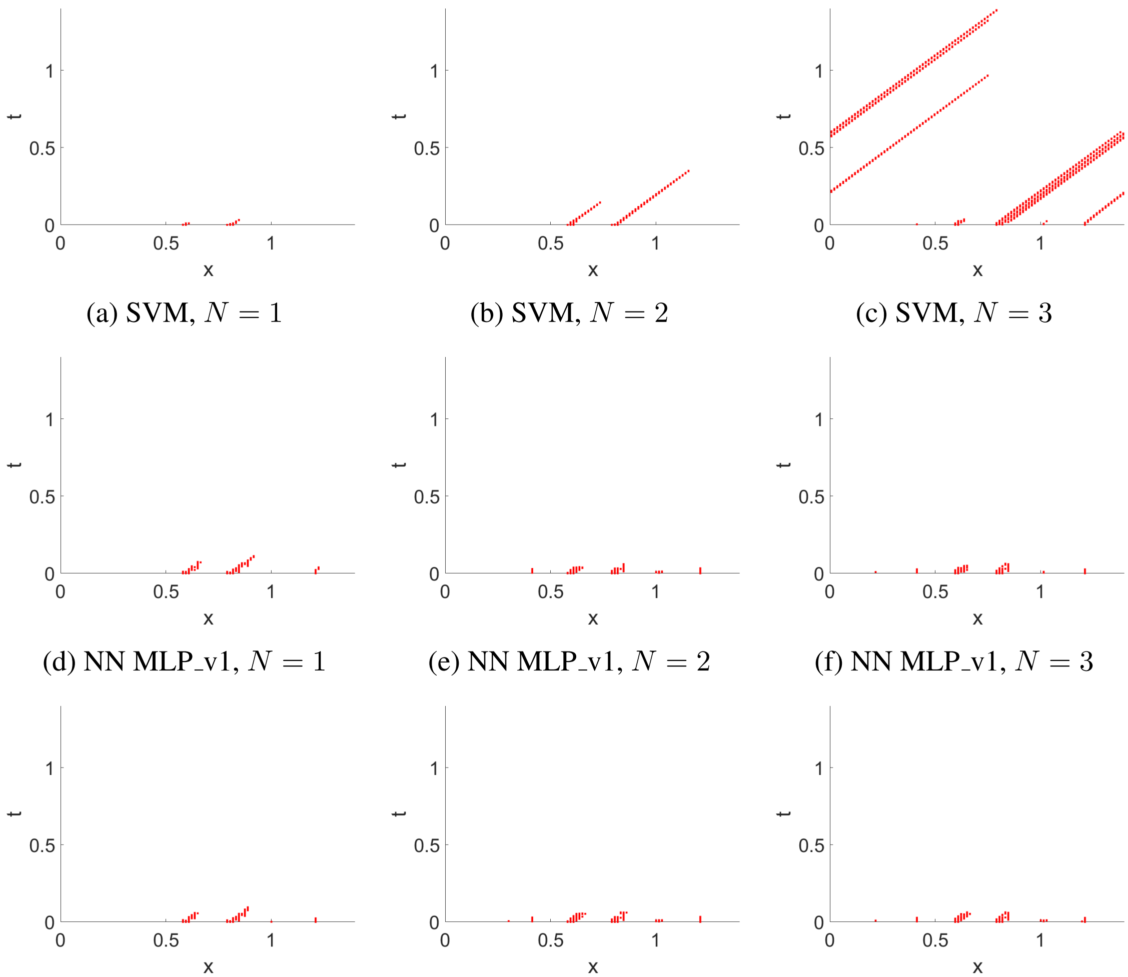

(a) Minmod-TVB, $N = 1$ (b) Minmod-TVB, $N = 2$ (c) Minmod-TVB, $N = 3$

(d) KXRCF, $N = 1$ (e) KXRCF, $N = 2$ (f) KXRCF, $N = 3$

(g) Fu and Shu, $N = 1$ (h) Fu and Shu, $N = 2$ (i) Fu and Shu, $N = 3$

(j) Vuik and Ryan TVB, $N = 1$ (k) Vuik and Ryan TVB, $N = 2$ (l) Vuik and Ryan TVB, $N = 3$

Figure 5.2: Troubled cell time histories for non-machine learning methods for the linear advection multi-wave problem.



(a) SVM, $N = 1$ (b) SVM, $N = 2$ (c) SVM, $N = 3$

(d) NN MLP_v1, $N = 1$ (e) NN MLP_v1, $N = 2$ (f) NN MLP_v1, $N = 3$

(g) NN MLP_v2, $N = 1$ (h) NN MLP_v2, $N = 2$ (i) NN MLP_v2, $N = 3$

Figure 5.3: Troubled cell time histories for the machine learning methods for the linear advection multi-wave problem.

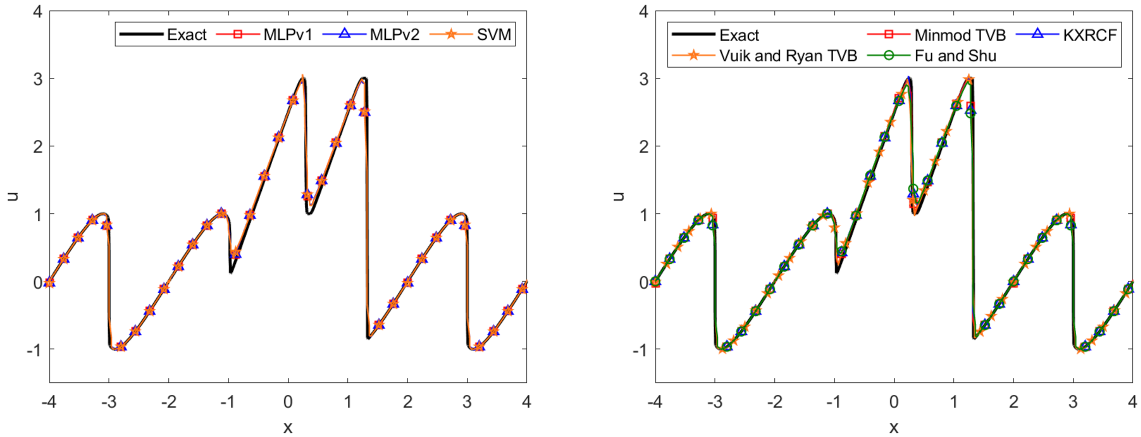
### 5.2.1 Compound Wave

For this test, the initial condition is composed of smooth and varying discontinuous sections:

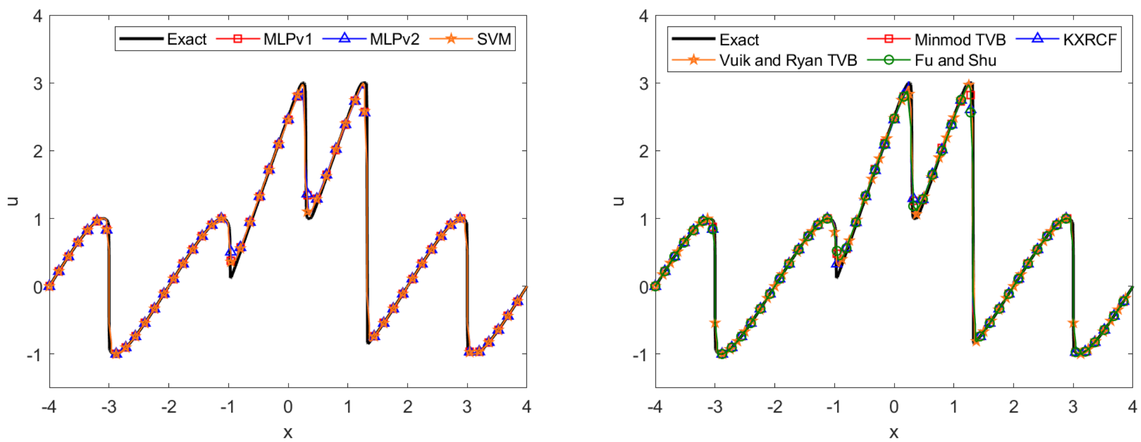
$$u_0(x) = \begin{cases} \sin(\pi x) & \text{if } |x| \geq 1, \\ 3 & \text{if } -1 \leq x < -0.5, \\ 1 & \text{if } -0.5 \leq x < 0, \\ 3 & \text{if } 0 \leq x < 0.5, \\ 2 & \text{if } 0.5 \leq x < 1 \end{cases} \tag{5.4}$$

on the domain $[0, 1]$ with periodic boundary conditions. The solution is simulated on a mesh of $K = 200$ cells until time $T = 0.4$, during which alternating shock and rarefaction waves develop.

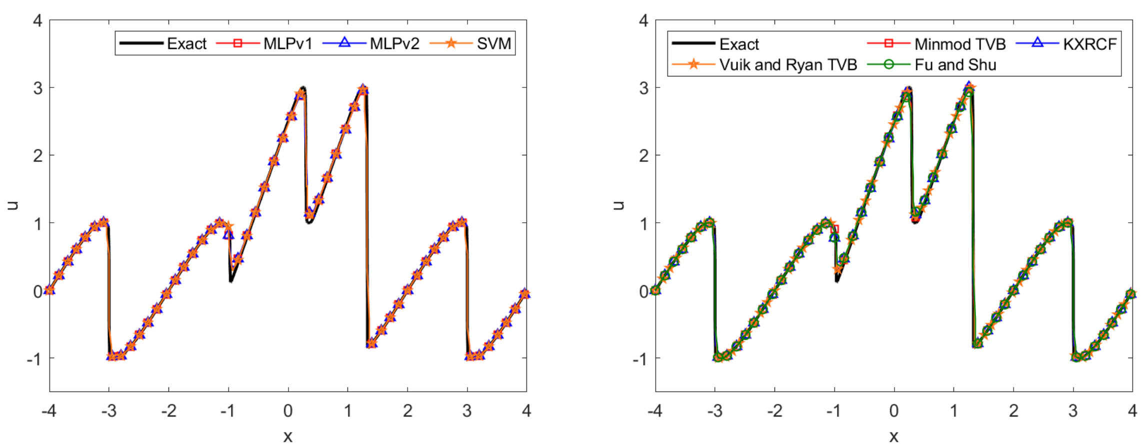
The SVM's numerical solutions for $N = 1, 2, 3$ are essentially indistinguishable from the neural networks' results and are competitive with the other indicators as well, as seen in Fig. 5.4. The SVM also indicates the same or slightly smaller numbers of troubled cells as the neural networks in Fig. 5.6. However, the minmod-TVB, KXRCF and Vuik and Ryan TVB indicators flag slightly less troubled cells than the SVM, as seen in Fig. 5.5.




(a) N=1

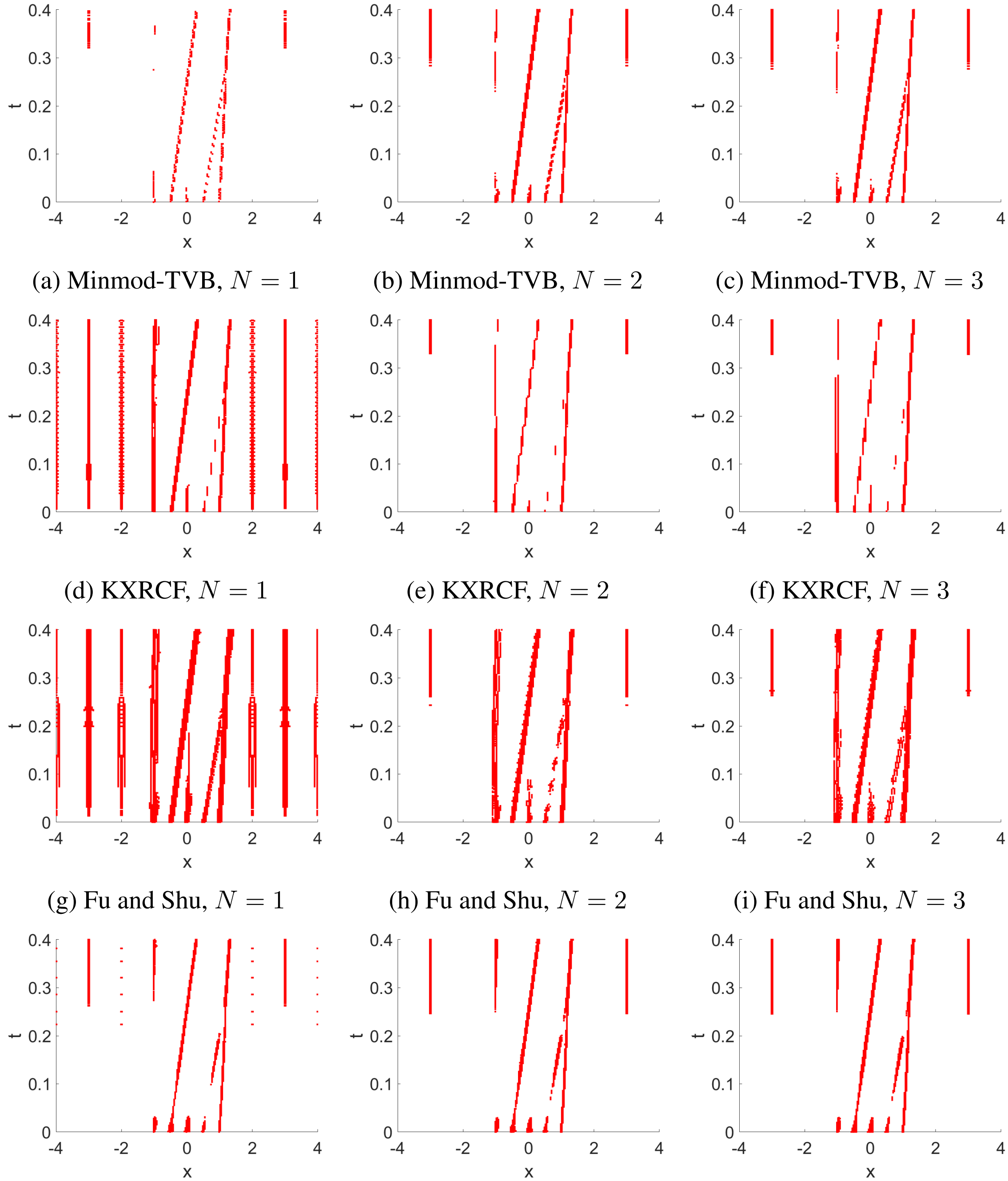



(b) N=2




(c) N=3

Figure 5.4: Solution of the compound wave problem for Burgers' equation, computed at $T = 0.4$ with $K = 200$. (a) degree $N = 1$ polynomial solution, (b) degree $N = 2$ polynomial solution, (c) degree $N = 3$ polynomial solution.



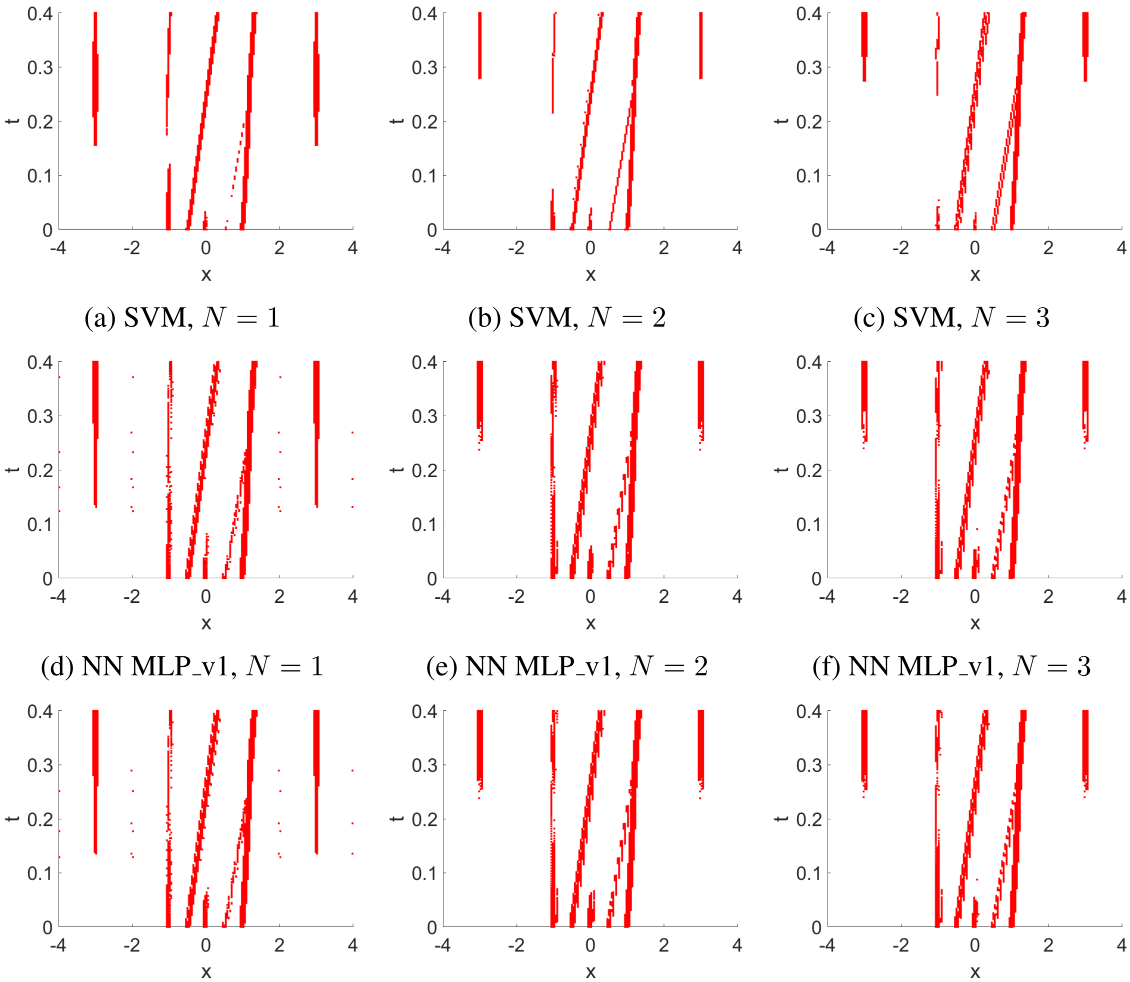

(a) Minmod-TVB, $N = 1$ (b) Minmod-TVB, $N = 2$ (c) Minmod-TVB, $N = 3$

(d) KXRCF, $N = 1$ (e) KXRCF, $N = 2$ (f) KXRCF, $N = 3$

(g) Fu and Shu, $N = 1$ (h) Fu and Shu, $N = 2$ (i) Fu and Shu, $N = 3$

(j) Vuik and Ryan TVB, $N = 1$ (k) Vuik and Ryan TVB, $N = 2$ (l) Vuik and Ryan TVB, $N = 3$

Figure 5.5: Troubled cell time histories for non-machine learning methods for the Burgers' equation compound wave problem.




(a) SVM, $N = 1$ (b) SVM, $N = 2$ (c) SVM, $N = 3$

(d) NN MLP_v1, $N = 1$ (e) NN MLP_v1, $N = 2$ (f) NN MLP_v1, $N = 3$

(g) NN MLP_v2, $N = 1$ (h) NN MLP_v2, $N = 2$ (i) NN MLP_v2, $N = 3$

Figure 5.6: Troubled cell time histories for the machine learning methods for the Burgers' equation compound wave problem.

## 5.3 Euler Equations

We consider the one dimensional Euler equations with $\rho$ as fluid density, $u$ as fluid velocity, and $p$ as fluid pressure:

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u\\ E\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho u\\ p+\rho u^2\\ (E+p)u\end{bmatrix}=0 \tag{5.5}$$

The total energy per unit volume $E$ is given by

$$E=\rho\left(\frac{u^2}{2}+e\right),$$

where $e$ is the specific internal energy given by an equation of state, $e = e(p, \rho)$. We chose the equation of state to be

$$e=\frac{p}{(\gamma-1)\rho},$$

where $\gamma = c_p/c_v$ is a ratio of specific heats; we set $\gamma = 1.4$ for all numerical tests.

For the Euler equations, one must choose the indicator-variables, the variables used in the indication of troubled cells, and the limited-variables, the variables used in the slope limiting aspect of RKDG methods. Similar to the choices made in [15], we choose the primitive variables $\rho, u, p$ as the indicator-variables, and we choose the local characteristic variables as the limited-variables to avoid spurious oscillations [12].

The CFL condition is CFL=0.4 for all following problems, and each problem has Dirichlet boundary conditions equal to initial conditions' values on the left and right sides. For each problem, we will show only the density solution and the troubled cell plots to conserve space.

**Remark 5.3.1.** *Troubled cell indicators used on the Euler equations can make the numerical method fail if loss of positivity of density occurs. For example, for the double rarefaction problem in Section 5.3.3, both TVB and KXRCF fail for $N = 1$, explaining why they do not have any parameters for this case in Table 5.1.*

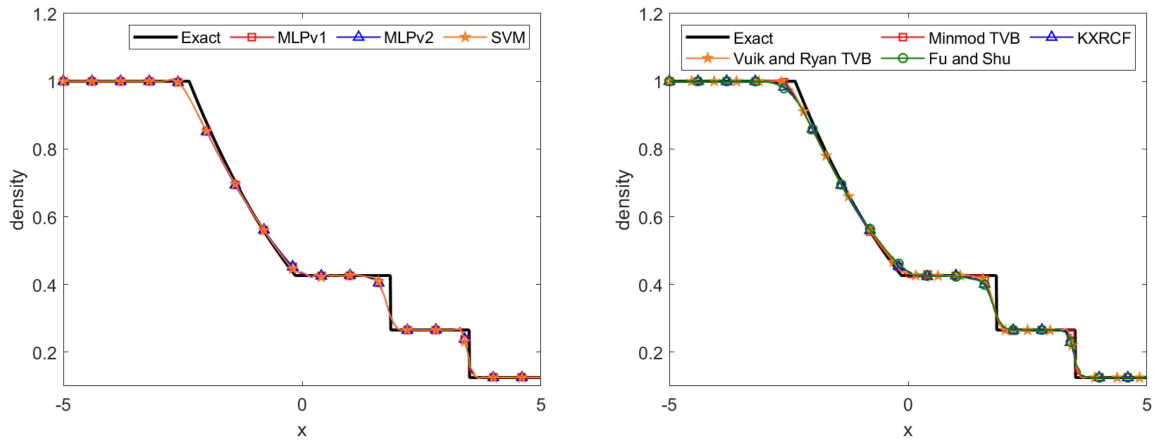
### 5.3.1 Sod Test

We test a mild shock tube proposed by Sod [20] with initial condition given by

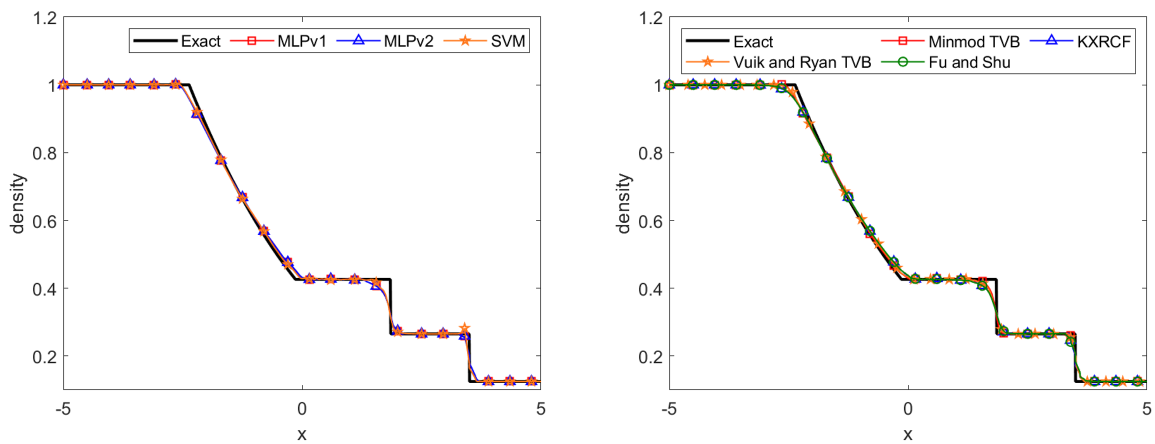
$$(\rho_0, u_0, p_0) = \begin{cases} (1, 0, 1) & \text{if } x < 0, \\ \\ (0.125, 0, 0.1) & \text{if } x \geq 0 \end{cases} \tag{5.6}$$

on the domain $[-1, 1]$. The solution is simulated on a mesh of $K = 100$ cells until $T = 2$.

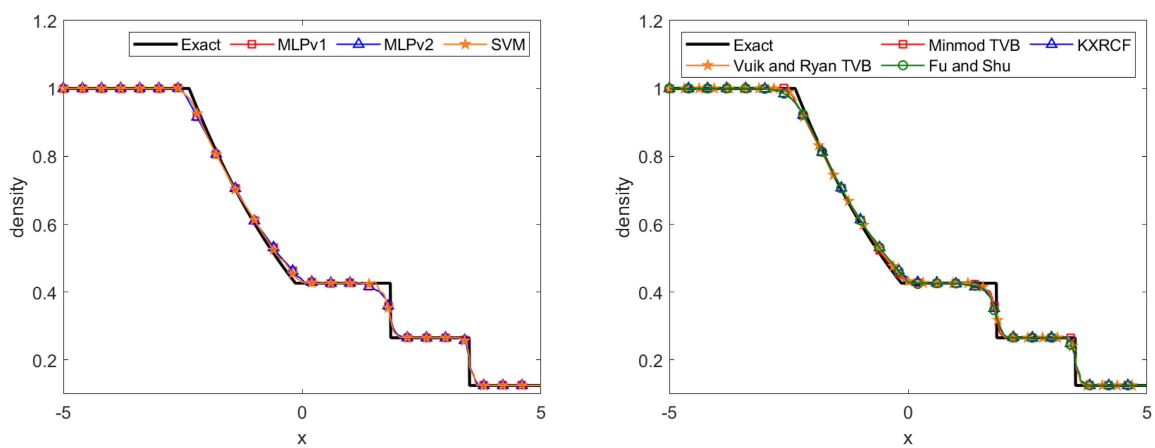
For the Sod problem, the SVM's numerical solutions are indistinguishable from the neural networks for $N = 1$, but seem to be slightly closer to the exact solution than the networks for $N = 2$ and $N = 3$, as seen in Fig. 5.7. These results are similar to the minmod-TVB and Vuik and Ryan TVB indicators, which are closer to the exact solution than KXRCF and Fu and Shu in this problem. The troubled cell time histories confirm than minmod-TVB and Vuik and Ryan TVB perform better than KXRCF and Fu and Shu, which have many more troubled cells (see Fig. 5.8). The SVM flags the least amount of troubled cells out of all the indicators in Fig. 5.9.




(a) N=1

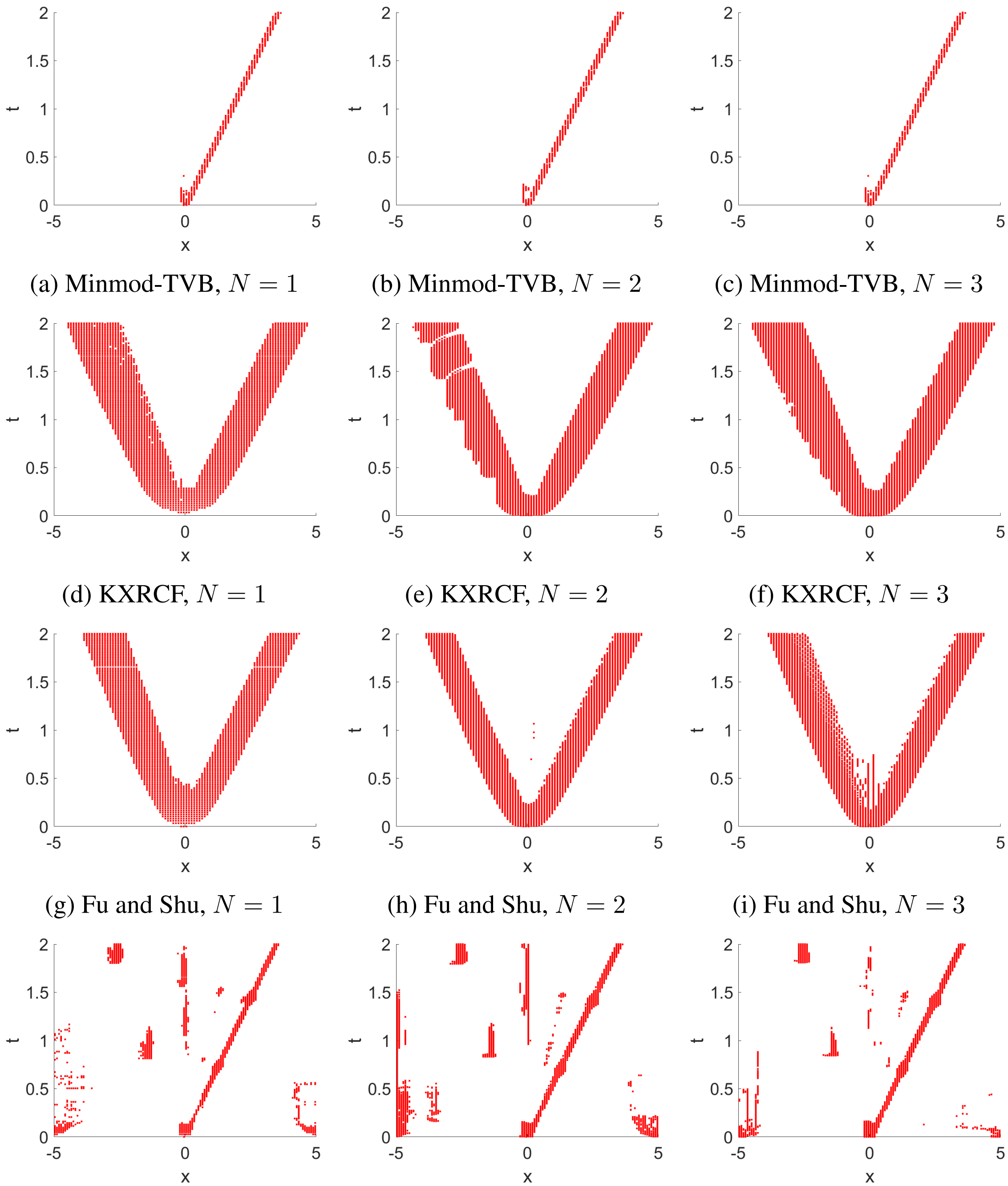



(b) N=2




(c) N=3

Figure 5.7: Density solutions of the Sod shock tube problem for the Euler equations, computed at $T = 2$ with $K = 100$. (a) degree $N = 1$ polynomial solution, (b) degree $N = 2$ polynomial solution, (c) degree $N = 3$ polynomial solution.



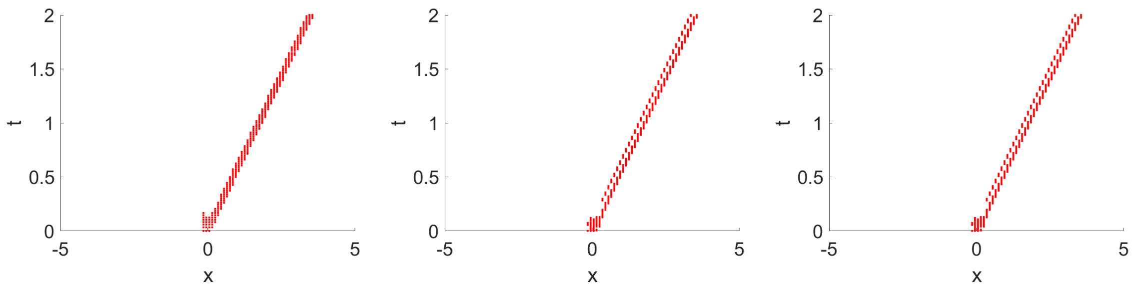

(a) Minmod-TVB, $N = 1$ (b) Minmod-TVB, $N = 2$ (c) Minmod-TVB, $N = 3$

(d) KXRCF, $N = 1$ (e) KXRCF, $N = 2$ (f) KXRCF, $N = 3$

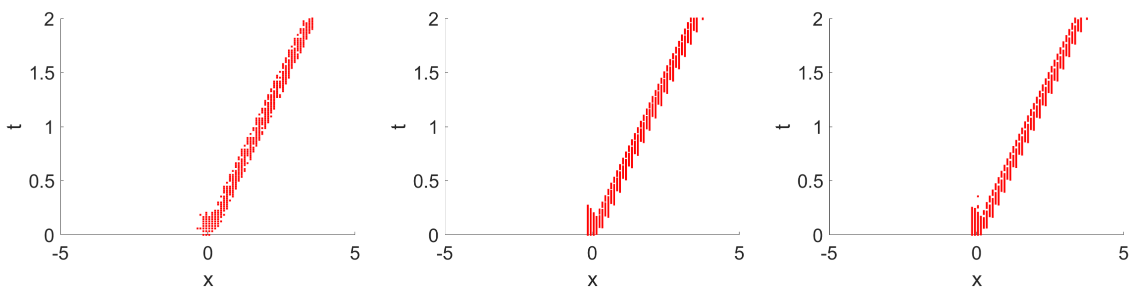
(g) Fu and Shu, $N = 1$ (h) Fu and Shu, $N = 2$ (i) Fu and Shu, $N = 3$

(j) Vuik and Ryan TVB, $N = 1$ (k) Vuik and Ryan TVB, $N = 2$ (l) Vuik and Ryan TVB, $N = 3$

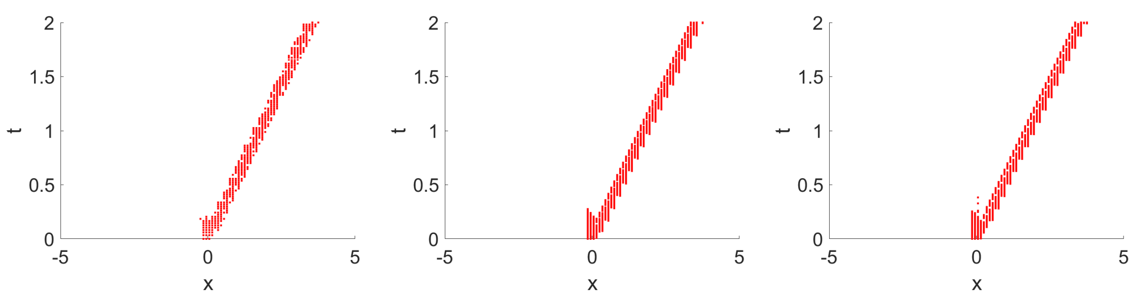
Figure 5.8: Troubled cell time histories for non-machine learning methods for the Sod shock tube Euler equations problem.




(a) SVM, $N = 1$ (b) SVM, $N = 2$ (c) SVM, $N = 3$




(d) NN MLP_v1, $N = 1$ (e) NN MLP_v1, $N = 2$ (f) NN MLP_v1, $N = 3$




(g) NN MLP_v2, $N = 1$ (h) NN MLP_v2, $N = 2$ (i) NN MLP_v2, $N = 3$

Figure 5.9: Troubled cell time histories for the machine learning methods for the Sod shock tube Euler equations problem.

### 5.3.2 Lax Test

We consider the Lax shock tube problem [9], with the initial condition given by

$$(\rho_0, u_0, p_0) = \begin{cases} (0.445, 0.698, 3.528) & \text{if } x < 0, \\ \\ (0.5, 0, 0.571) & \text{if } x \geq 0, \end{cases} \tag{5.7}$$

on the domain $[-5, 5]$. The solution is simulated on a mesh of $K = 200$ cells until $T = 1.3$.

For the Lax problem, the SVM's solutions are almost indistinguishable from the neural networks' solutions, except for a slightly higher oscillation on the right side of the shock wave, seen in Fig. 5.10 (a) and (b). The SVM also flags about the same amount of troubled cells as the neural networks, while the minmod-TVB indicator for $N = 3$ flags the least amount of troubled cells in Fig. 5.12 and Fig. 5.11.

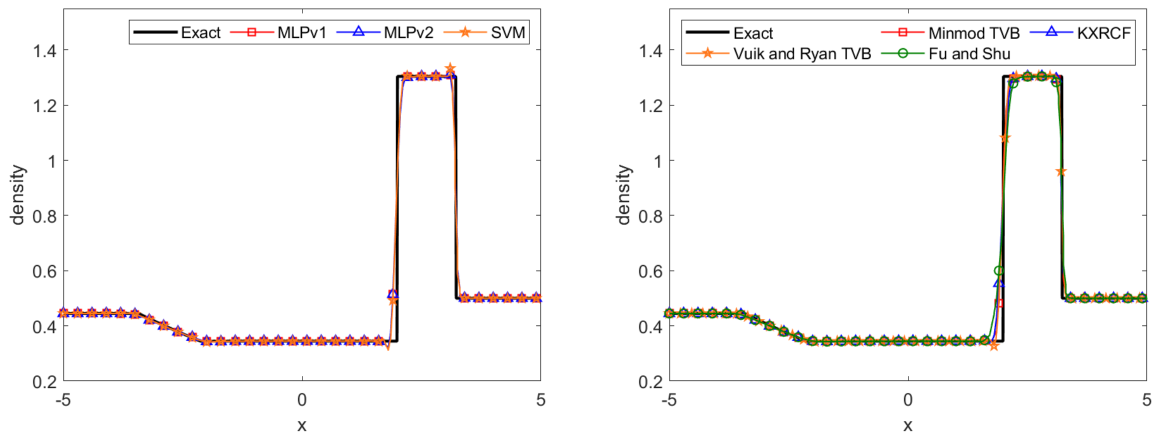
### 5.3.3 Double Rarefaction

In this test, we consider the Riemann problem with the initial condition from [11]:

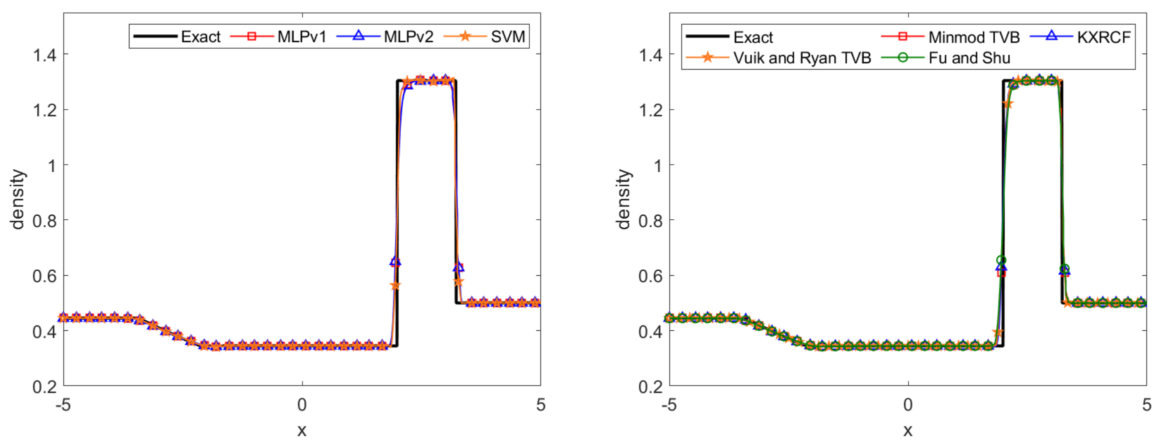
$$(\rho_0, u_0, p_0) = \begin{cases} (7, -1, 0.2) & \text{if } x < 0, \\ \\ (7, 1, 0.2) & \text{if } x \geq 0 \end{cases} \tag{5.8}$$

for $x \in [-1, 1]$. This test is simulated on a mesh with $K = 128$ cells until the time $T = 0.6$. The solution is composed of two rarefaction waves that pull in opposite directions, causing a near-vacuum region to appear at $x = 0$ [11].

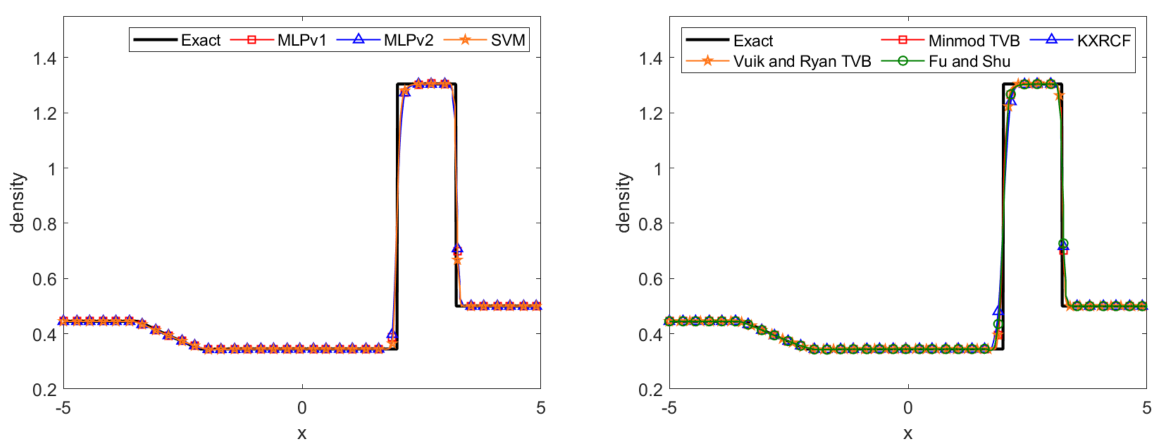
For $N = 1$, the minmod-TVB and KXRCF indicators result in positivity loss for all tried




(a) N=1

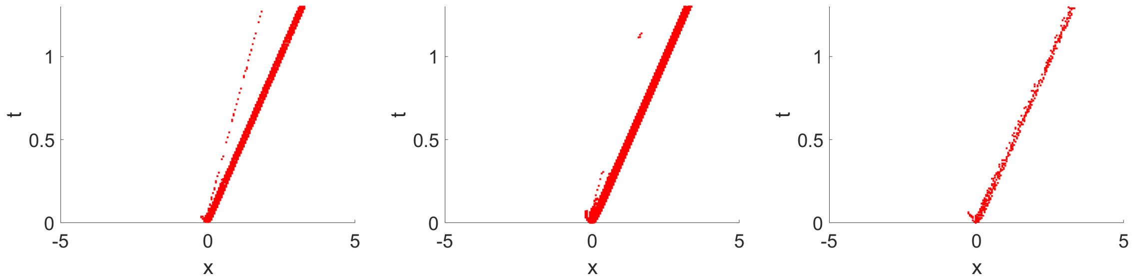



(b) N=2

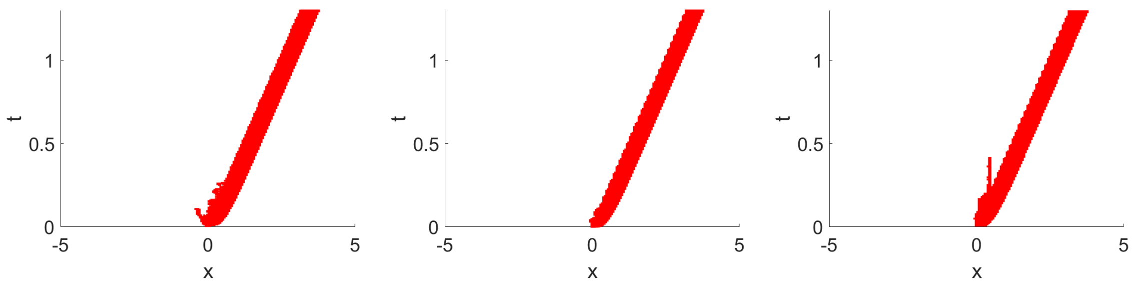



(c) N=3

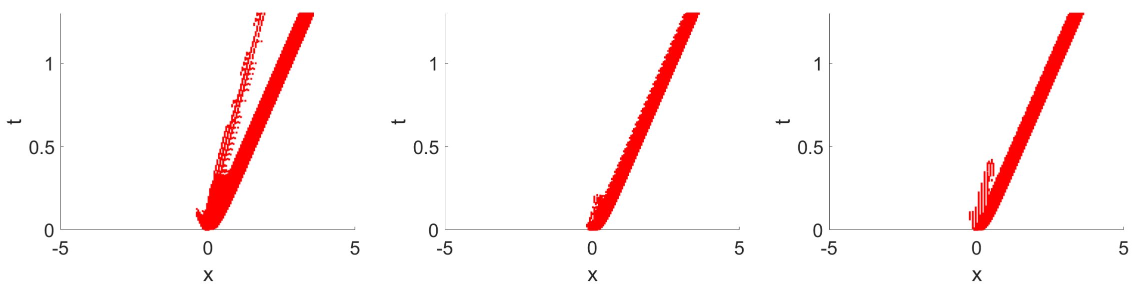
Figure 5.10: Density solutions of the Lax shock tube problem for the Euler equations, computed at $T = 1.3$ with $K = 200$. (a) degree $N = 1$ polynomial solution, (b) degree $N = 2$ polynomial solution, (c) degree $N = 3$ polynomial solution.



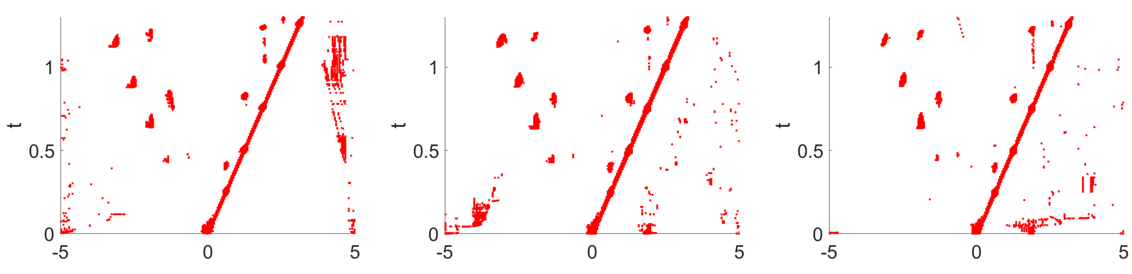

(a) Minmod-TVB, $N = 1$ (b) Minmod-TVB, $N = 2$ (c) Minmod-TVB, $N = 3$




(d) KXRCF, $N = 1$ (e) KXRCF, $N = 2$ (f) KXRCF, $N = 3$

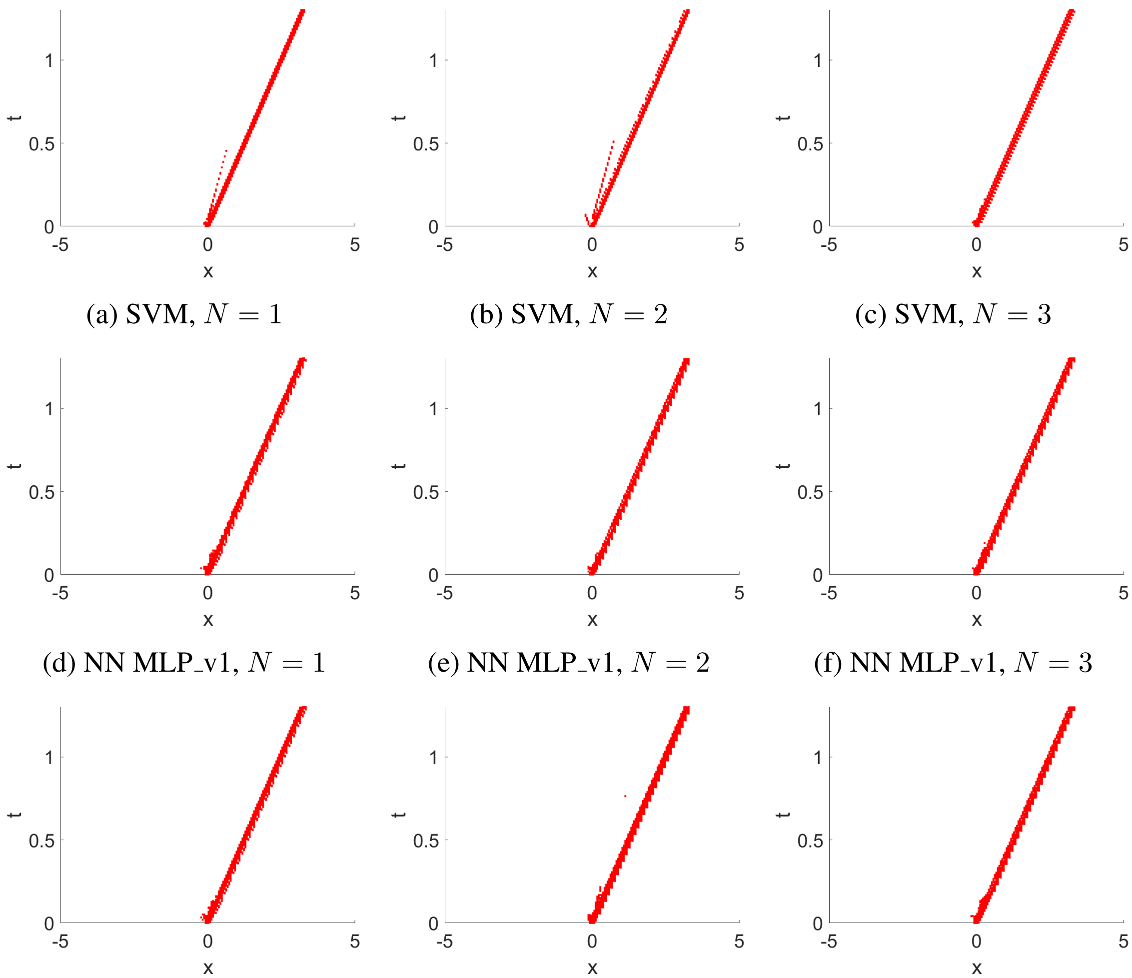



(g) Fu and Shu, $N = 1$ (h) Fu and Shu, $N = 2$ (i) Fu and Shu, $N = 3$




(j) Vuik and Ryan TVB, $N = 1$ (k) Vuik and Ryan TVB, $N = 2$ (l) Vuik and Ryan TVB, $N = 3$

Figure 5.11: Troubled cell time histories for non-machine learning methods for the Lax shock tube problem for the Euler equations.

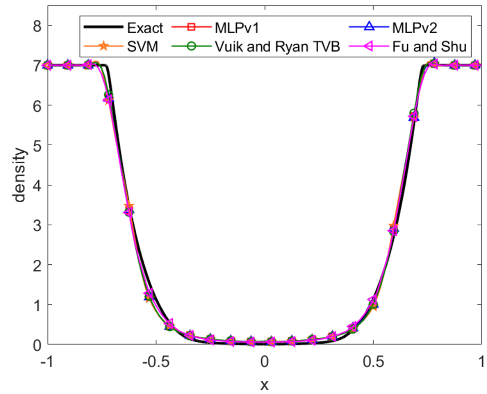


(a) SVM, $N = 1$ (b) SVM, $N = 2$ (c) SVM, $N = 3$

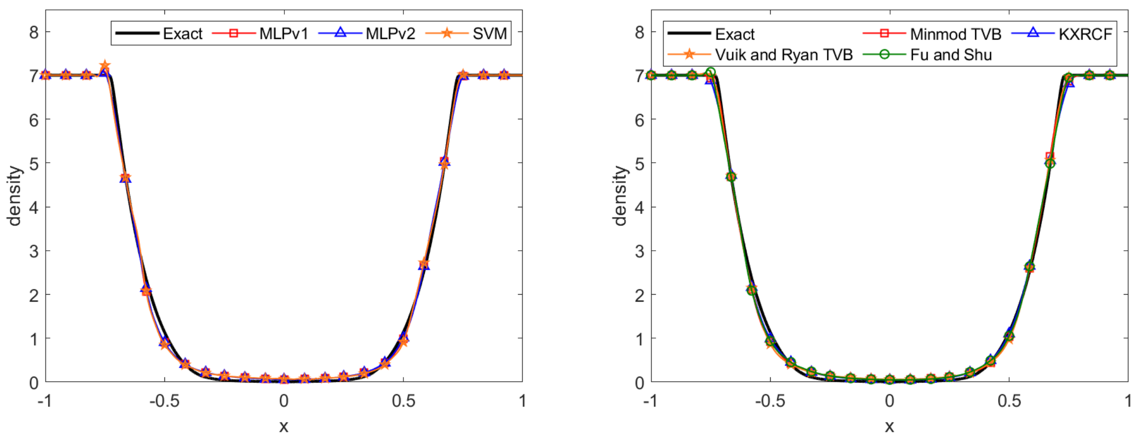
(d) NN MLP_v1, $N = 1$ (e) NN MLP_v1, $N = 2$ (f) NN MLP_v1, $N = 3$

(g) NN MLP_v2, $N = 1$ (h) NN MLP_v2, $N = 2$ (i) NN MLP_v2, $N = 3$

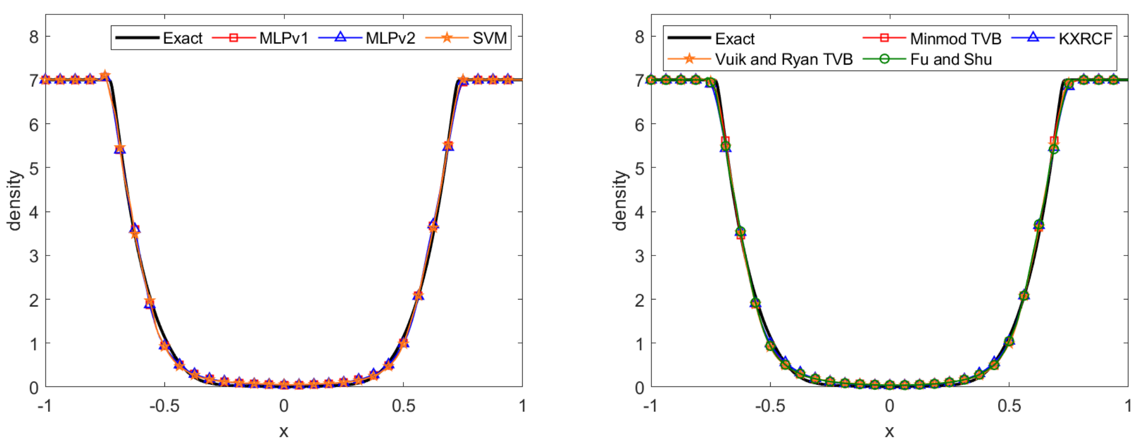
Figure 5.12: Troubled cell time histories for the machine learning methods for the Lax shock tube problem for the Euler equations.




(a) N=1

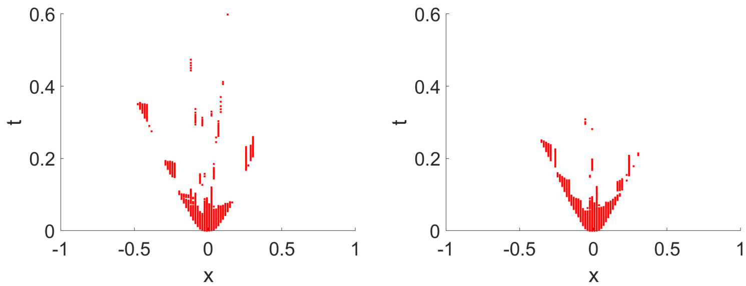



(b) N=2

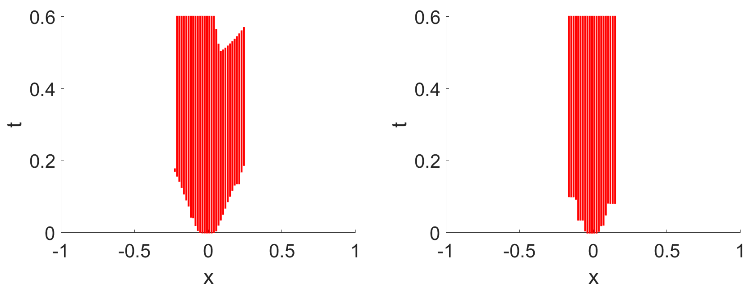



(c) N=3

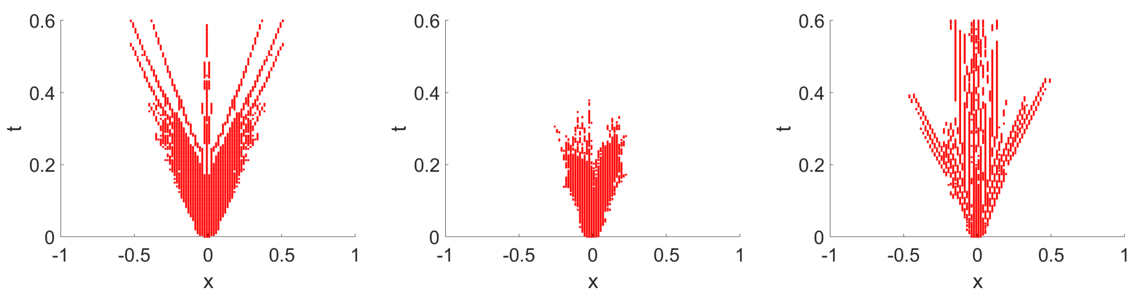
Figure 5.13: Density solutions of the double rarefaction problem for the Euler equations, computed at $T = 0.6$ with $K = 128$. (a) degree $N = 1$ polynomial solution, (b) degree $N = 2$ polynomial solution, (c) degree $N = 3$ polynomial solution.



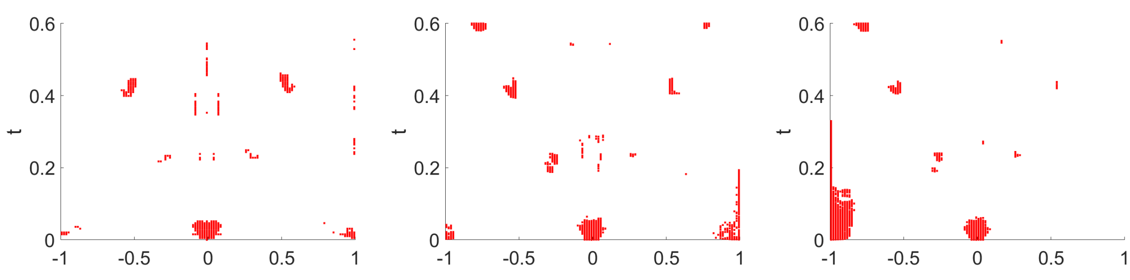
(a) Minmod-TVB, $N = 2$ (b) Minmod-TVB, $N = 3$



(c) KXRCF, $N = 2$ (d) KXRCF, $N = 3$

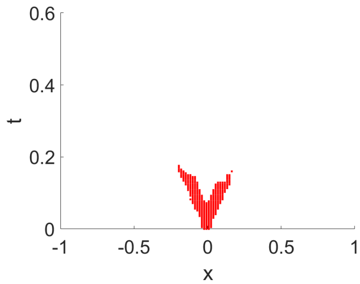


(e) Fu and Shu, $N = 1$ (f) Fu and Shu, $N = 2$ (g) Fu and Shu, $N = 3$

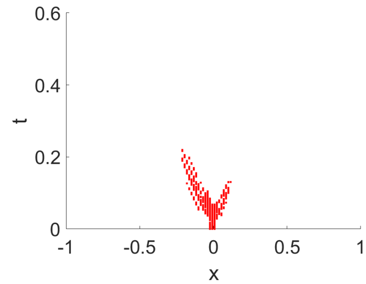


(h) Vuik and Ryan TVB, $N = 1$ (i) Vuik and Ryan TVB, $N = 2$ (j) Vuik and Ryan TVB, $N = 3$

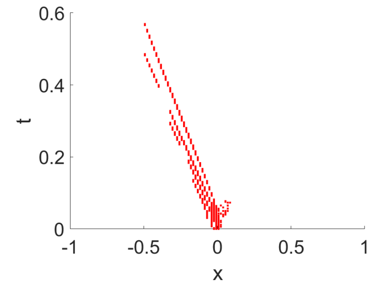
Figure 5.14: Troubled cell time histories for non-machine learning methods for the double rarefaction problem for the Euler equations.



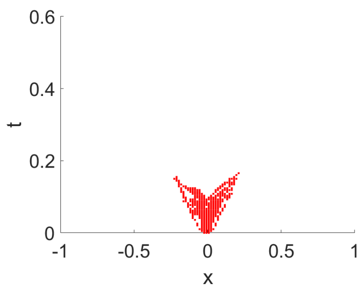

(a) SVM, $N = 1$



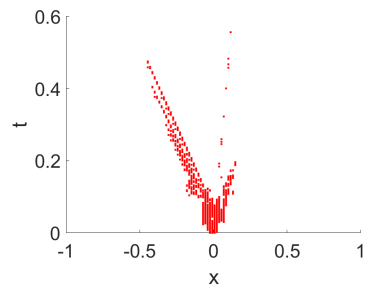

(b) SVM, $N = 2$



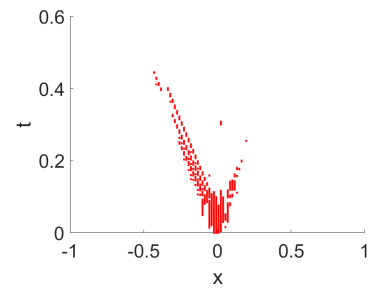

(c) SVM, $N = 3$



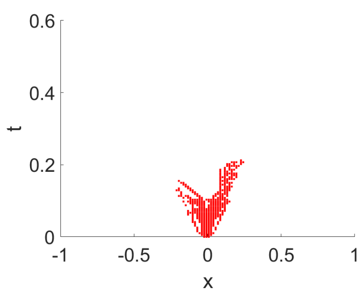

(d) NN MLP_v1, $N = 1$



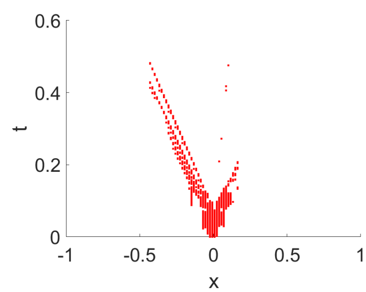

(e) NN MLP_v1, $N = 2$



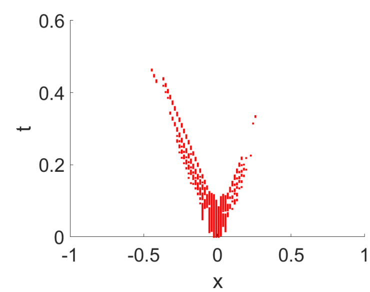

(f) NN MLP_v1, $N = 3$




(g) NN MLP_v2, $N = 1$




(h) NN MLP_v2, $N = 2$




(i) NN MLP_v2, $N = 3$

Figure 5.15: Troubled cell time histories for the machine learning methods for the double rarefaction for the Euler equations.

parameters. Once again, the SVM performs well, on par with the neural networks and the other troubled cell indicators seen in Fig. 5.13. Fig. 5.15 and Fig. 5.14 show that the SVM has the least amount of troubled cells among the neural networks, Fu and Shu, and KXRCF. The minmod-TVB indicator shows a minimal amount as well, but does not work for $N = 1$. The Vuik and Ryan TVB indicator also flags a minimal amount of troubled cells, but interestingly, the shape is much different from the other indicators.

### 5.3.4 Additional Numerical Tests

The SVM was also tested on two other problems for the Euler equations. The first is the shock-entropy problem proposed by Shu and Osher [19], which describes the interaction of a right moving shock with a high-frequency oscillatory smooth wave. Unfortunately, the SVM received positivity loss errors for all degrees of $N$, while the neural networks perform well for any $N$ value. These errors demonstrate that the SVM is not flagging the right troubled cells; thus, the lack of limiting leads to spurious oscillations, triggering the positivity loss error.

The second problem is the left half of the blast-wave problem from [22], where a severe shock is applied to the system. Interestingly, the SVM for $N = 1$ and $N = 2$ ran successfully, but the $N = 3$ SVM case had a positivity loss error. Referring to Table 4.4, we note that the $N = 3$ SVM has different parameters from the other two SVMs. This difference requires further investigation to hopefully allow the SVM to generalize to these two additional Euler problems.

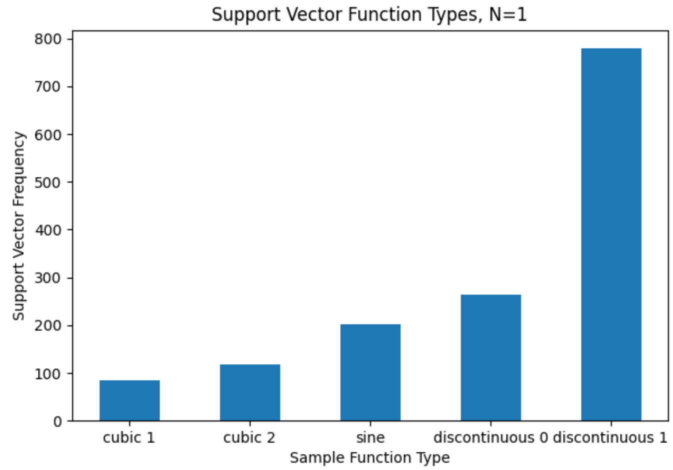
## 5.4 Analyzing the SVM indicator

In order to analyze the SVM with more than just numerical simulations, we study the nature of the support vectors that are used in the SVM's decision function. We also perform a sensitivity analysis on the SVM and two neural networks, evaluating each model's robustness to noise.

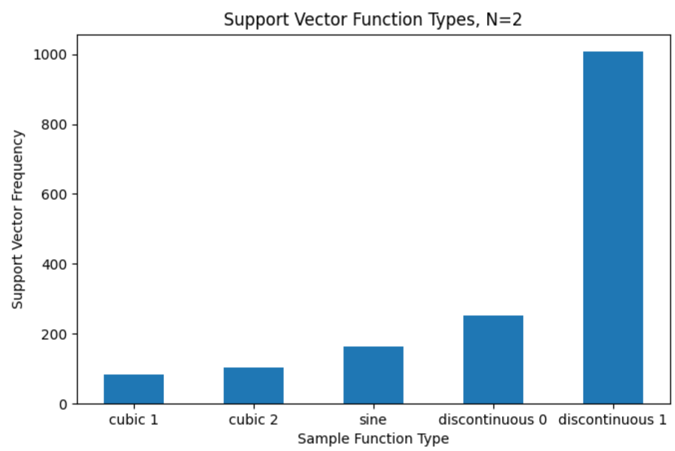
### 5.4.1 Support Vector Analysis

The goal of this analysis is to evaluate which functions are most important in determining the support vectors, which may provide some insights on how to build a data set for training SVMs, especially in higher dimensions. We create histograms of the frequency of each type of training function present in the set of support vectors in Fig. 5.16. Note that we split the discontinuous category of functions into those labeled with zeros or ones. The smooth cubic and sine functions are always labeled with zeros.

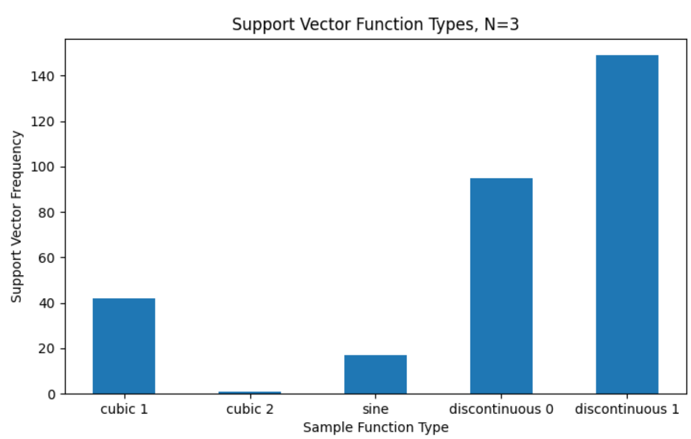
From these histograms, it is clear that more of the support vectors come from the discontinuous functions labeled with 1. For $N = 1$ and $N = 2$, the sum of the counts of the label-0 functions is still less than the number of discontinuous-1 functions, so the label-1 information is more important to the SVM decision function. It is interesting that even the discontinuous-0 functions have the highest frequency as support vectors among the label-0 functions. For $N = 1$ and $N = 2$, the sine functions seem to be more important as support vectors than each of the cubic function types. However, though the graphs for $N = 1$ and $N = 2$ look quite similar, the frequency of the cubic 2 and sine types are much smaller for the $N = 3$ case than the others. In addition, the total amount of label-0 functions in the support vectors is larger than the amount of discontinuous-1 functions. These discrepancies may also be related to the difference




(a)




(b)




(c)

Figure 5.16: Support vector histograms for each degree of SVM.

in parameters for $N = 3$ and its inability to resolve the left blast wave problem.

Overall, the discontinuous functions are the most important part of the data set to the SVM. This support vector information can serve as a guide to constructing more refined training sets for other ML approaches to troubled cell indication.

### 5.4.2 Sensitivity Analysis

Since the SVM and neural networks are very similar in performance and troubled cell indication, we investigate their robustness to noise to further compare the two types of models. Let us define the input data set to the SVM as $X$. We take a standard normal array $\eta$ and set $X \to X + \varepsilon\eta$, where the scaling factor $\varepsilon = \alpha|\bar{X}|$. In other words, we take a certain fraction $\alpha$ of the mean of the absolute value of the input data set, and use it to scale the standard Gaussian noise vector $\eta$. So, if $\alpha = 0.05$, this method will add about 5% noise to the input data set. Then, we test this noisy data in the SVM and compare the output to the original labels. If the models are more sensitive to noise, they will have lower accuracy on the noisy data.

| $\alpha$ | SVM Accuracy | MLP_v1 Accuracy | MLP_v2 Accuracy |
|---|---|---|---|
| 0.05 | 0.6265 | 0.4338 | 0.4245 |
| 0.04 | 0.7344 | 0.4789 | 0.4775 |
| 0.03 | 0.8864 | 0.5397 | 0.5369 |
| 0.02 | 0.9741 | 0.6283 | 0.6279 |
| 0.01 | 0.9815 | 0.7853 | 0.7945 |

Table 5.4: Accuracy values of ML models on perturbed input data for different alpha values.

For the SVM, we test a variety of $\alpha$ values and list their accuracy scores in Table 5.4. Then, we test the neural networks in the same way using the perturbed SVM input data. After converting the SVM input data to the inputs for the neural network, MLP_v1 and MLP_v2 both had about 96% accuracy on the non-perturbed data set. The accuracies for both versions of the network are also listed in Table 5.4.

It is clear that the neural networks are much more sensitive to noise than the SVM; higher values of $\alpha$ result in much lower accuracies for the neural networks.

# Chapter 6: Conclusions and Future Work

This thesis demonstrates that support vector machines as troubled cell indicators are a reasonable alternative to neural networks, while decision trees did not perform well enough to be considered as a troubled cell indicator. SVMs provided almost identical results to the neural networks in several numerical tests and generally flagged the about same amount or less troubled cells than other troubled cell indicators. However, the SVM did fail for the $N = 3$ case of the left blast wave problem and failed completely for the shock entropy problem. Several improvements could be made to the SVM that may fix these failures, such as increasing the $\varepsilon$ range outside the target cell's boundaries that flags a troubled cell (currently, we use the value $\varepsilon = 1/2$). In addition, since the sensitivity analysis in Chapter 5 concludes that the SVM is more robust to noise, the SVM could be improved by retraining with noise added to half of the original data set.

Analysis of the SVM support vectors provided insight into the relative importance of training data function types in determining the troubled cell decision function. The training set's discontinuous functions formed the majority of the support vectors, even when the discontinuity was located outside of the target cell. And, rather than the individual cubic functions, the SVM preferred to use the sine functions for the smooth data support vectors. Information from the support vector analysis can be used to further improve both SVM and neural network troubled cell indicator methods. For example, the SVM could be retrained with higher-frequency sine functions or discontinuous functions that resemble the shock entropy solution, which might help the SVM flag the correct cells for limiting and avoid positivity loss errors. Also, since the SVM has demonstrated the importance of the support vectors in the troubled cell decision function, a neural network could train on the SVM's support vectors to possibly improve its performance.

The neural networks and SVMs in this thesis share a nonlinear nature that allows them to learn the troubled cell indication function better than a decision tree. Though the SVM failed in some cases and needs some improvement, the process of training a neural network is more complicated than training a SVM; so, SVMs may provide a simpler yet just as accurate way to use machine learning for troubled cell indication. However, the SVM is specialized for each degree of the polynomial solution because it receives the nodal coefficients as inputs, but the same neural network can work on different degrees. It may then also be worth comparing the SVM with nodal coefficients to an SVM with the same inputs as the neural networks that generalizes to several degrees. Perhaps the current nodal coefficient inputs are causing the SVM to fail for the two Euler tests in Section 5.3.4. The MLP_v1 and MLP_v2 networks have also been trained to deal with perturbed, non-uniform meshes, so training and evaluating the SVM on perturbed meshes may provide another point of comparison between the two ML methods.

By testing different machine learning techniques against traditional troubled cell indicators, we have demonstrated that SVMs can be an effective and accurate troubled cell indicator. Training an SVM only once allows its application to a wide array of conservation law problems, giving SVMs an advantage over traditional troubled cell indicators that rely on varying, problem-dependent parameters. And, given the success of the SVM's application in the context of RKDG schemes for conservation laws, SVMs may be successful in other PDE scientific machine learning contexts involving classification or parameter estimation.

# Bibliography


[1] Bernardo Cockburn and Chi-Wang Shu. Tvb runge-kutta local projection discontinuous galerkin finite element method for conservation laws ii: General framework. *Mathematics of Computation*, 52(186):411–435, 1989.

[2] Bernardo Cockburn and Chi-Wang Shu. Runge-kutta discontinuous galerkin methods for convection-dominated problems. *Journal of Scientific Computing*, 16(3), 2001.

[3] Niccolo Discacciati, Jan S. Hesthaven, and Deep Ray. Controlling oscillations in high-order discontinuous galerkin schemes using artificial viscosity tuned by neural networks. *Journal of Computational Physics*, 409:109304, 2020.

[4] Guosheng Fu and Chi-Wang Shu. A new troubled-cell indicator for discontinuous galerkin methods for hyperbolic conservation laws. *Journal of Computational Physics*, 347:305–327, 2017.

[5] Jan Hesthaven. *Spectral Methods for Hyperbolic Problems*, volume 17, chapter 17, pages 441–466. North Holland, 11 2016.

[6] Jan S. Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin Methods*. Springer, 2008.

[7] S.R. Siva Prasad Kochi and M. Ramakrishna. A review of troubled cell indicators for discontinuous galerkin method on structured and unstructured meshes. under submission, 2024.

[8] L. Krivodonova, J. Xin, J.-F. Remacle, N. Chevaugeon, and J.E. Flaherty. Shock detection and limiting with discontinuous galerkin methods for hyperbolic conservation laws. *Applied Numerical Mathematics*, 48:323–338, 2004.

[9] Peter D. Lax. Weak solutions of nonlinear hyperbolic equations and their numerical computation. *Communications on Pure and Applied Mathematics*, 7:159–193, 1954.

[10] Randall J. LeVeque. *Finite-Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2004.

[11] Timur Linde and Philip Roe. Robust euler codes. In *13th computational fluid dynamics conference*, page 2098, 1997.

[12] Jianxian Qiu and Chi-Wang Shu. On the construction, comparison, and local characteristic decomposition for high-order central weno schemes. *Journal of Computational Physics*, 183:187–209, 2002.

[13] Jianxian Qiu and Chi-Wang Shu. A comparison of troubled-cell indicators for runge-kutta discontinuous galerkin methods using weighted essentially nonoscillatory limiters. *SIAM Journal on Scientific Computing*, 27(3):995–1013, 2005.

[14] Deep Ray and Philip Charles. Learning weno for entropy stable schemes to solve conservation laws. under submission, 2024.

[15] Deep Ray and Jan Hesthaven. An artificial neural network as a troubled-cell indicator. *Journal of Computational Physics*, 367:166–191, 2018.

[16] Deep Ray and Jan Hesthaven. Detecting troubled-cells on two-dimensional unstructured grids using a neural network. *Journal of Computational Physics*, 397, 2019.

[17] Lukas Schwander, Deep Ray, and Jan Hesthaven. Controlling oscillations in spectral methods by local artificial viscosity governed by neural networks. *Journal of Computational Physics*, 431, 2021.

[18] Chi-Wang Shu. Tvb uniformly high-order schemes for conservation laws. *Mathematics of Computation*, 49(179):105–121, 1987.

[19] Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes, ii. *Journal of Computational Physics*, 83:32–78, 1989.

[20] Gary A. Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27(1):1–31, 1978.

[21] M. J. Vuik and J. K. Ryan. Automated parameters for troubled-cell indicators using outlier detection. *SIAM Journal on Scientific Computing*, 38(1):A84–A104, 2016.

[22] Paul Woodward and Phillip Colella. The numerical simulation of two-dimensional fluid flow with strong shocks. *Journal of Computational Physics*, 54:115–173, 1984.

[23] Jian Yu and Jan S. Hesthaven. A study of several artificial viscosity models within the discontinuous galerkin framework. *Communications in Computational Physics*, 27(5):1309–1343, 2020.

[24] Xinghui Zhong and Chi-Wang Shu. A simple weighted essentially nonoscillatory limiter for runge-kutta discontinuous galerkin methods. *Journal of Computational Physics*, 232:397–415, 2012.