Modeling Turbulence in Tokamak Plasmas using Reservoir Computing

Nathaniel Barbour
Advisor: Professor Bill Dorland
Department of Physics, University of Maryland

May 13, 2020

1 Background

Turbulence in tokamaks, a geometric configuration for magnetic plasma fusion experiments, is a significant catalyst for driving the radial transport of heat. To increase the efficiency of magnetic fusion experiments, radial heat flux must be mitigated. When radial heat flux increases, additional energy must be introduced into the system via external heating in order to sustain the core temperatures required for fusion reactions to occur. Additionally, there exists an experimentally observed but not fully theoretically understood transition to a higher-confinement regime called H-mode which permits lower amounts of turbulence. This H-mode also appears to have a limit beyond which confinement does not significantly improve. One of the grand challenges for plasma theorists is to develop a collective understanding of what plasma parameters, experimental parameters, and magnetic field configurations lead to a sustained H-mode plasma. While experiments both are and can be constructed to probe this question by trial and error, tokamaks are large, expensive machines that require extensive planning stages. As a demonstrative example, ITER, the major international fusion experiment currently under construction in southern France, is designed to be both the most powerful and efficient tokamak to date, crossing the elusive ignition threshold and generating 500 MW of fusion power with only 50 MW of input power. ITER is an expansive international project that has required over three decades of planning and billions of dollars of funding. Developing accurate, efficient models of how plasma performance responds to changes in experimental parameters and geometries can save significant expenditures of time and money by identifying the design considerations that are most conducive to H-mode operation.

Modeling radial transport and understanding the transition from low-confinement to high-confinement are inherently multiscale problems, as they require determining coefficients for diffusion that occurs on a time scale that is slow with respect to the time scale characteristic to the turbulence that drives the diffusion. To that end, Professor Dorland has developed a multiscale gyrokinetic simulation code, GS2, to study the fast turbulent dynamics, but the bottleneck to rapidly studying the myriad of compelling physics questions that arise is the computational expense of the simulations. With GX, a pseudospectral code which has been benchmarked as a successor to GS2 but not yet published, significant progress has been made toward achieving faster turbulence solutions using conventional algorithm improvements and utilizing basis functions that are conducive to more efficient computation. However, there may be an opportunity to improve upon those methods even further by leveraging advances in the rapidly expanding field of machine learning.

To model the radial transport in practice, turbulence codes like GS2 [1] and GX solve the equations governing the turbulent dynamics, calculate the turbulent heat flux, Q, an averaged and integrated quantity defined in Appendix H of N.R. Mandell et al. [2]. 1-D transport solvers like Trinity [3] then solve for radial transport. While the process is deterministic in the sense that a given gyrokinetic distribution function, g, will eventually map to a specific turbulent heat flux Q, there is no known transfer function from the gyrokinetic distribution function calculated by turbulence codes to the final turbulent heat flux. In principle, it should be possible for machine learning to solve this nonlinear problem, and we intend to explore that possibility in this work. If it is possible to map the distribution function to a turbulent heat flux at a later point in time without directly solving for the underlying turbulent dynamics at high temporal resolution, then the impact of plasma parameters on turbulent dynamics can be calculated significantly more efficiently. However, we first must demonstrate the successful implementation of a machine learning algorithm for the solution of a simpler PDE. For this problem, we choose the 1-D Kuramoto-Sivashinsky equation.

1.1 1-D Kuramoto-Sivashinsky Equation

The 1-D Kuramoto-Sivashinsky (KS) equation is the one-dimensional nonlinear differential equation:

$$\frac{\partial u}{\partial t} + \frac{\partial^4 u}{\partial x^4} + \frac{\partial^2 u}{\partial x^2} + u\frac{\partial u}{\partial x} = 0$$

The solution, $u(x,t)$, is periodic on the interval [0,L). Like in the turbulent system of interest for the full physics problem, there is a quadratic nonlinear term, $u\frac{\partial u}{\partial x}$. Dissipation in the equation stems from the fourth derivative term. However, one of the characteristic aspects of the equation is the anti-diffusion effect of the $\frac{\partial^2 u}{\partial x^2}$ term. These effects contribute to nontrivial chaotic dynamics in both time and space. Also of note is that the KS equation arises in plasma physics when analyzing trapped ion mode instabilities: in 1975, LaQuey et al. derived a form of the 1-D KS equation for trapped ion modes, where the solution represents a normalized form of the electrostatic potential [4]. They subsequently presented, but did not derive, a diffusion coefficient for particle transport in the plasma that scales with $(\frac{\partial u}{\partial x})^2$ averaged over space. This diffusion coefficient determines the rate of radial particle flux and will be a function of interest for our analysis of the 1-D KS equation. A typical solution for the KS equation is presented later in Figure 2.

1.2 Machine Learning: Neural Networks

In recent years, machine learning techniques have been employed in a plethora of domains, including classification, pattern-recognition, and predictive modeling of dynamical systems. In broad terms, machine learning techniques allow predictive algorithms to improve their predictive capabilities through an iterative "training" process. One common machine learning approach is the artificial neural network. The basic structure of a traditional artificial neural network is a feed-forward structure. Input data is fed to a layer of neuron-like units. These neurons apply a nonlinear transformation to the weighted input data and pass outputs to each successive layer until the final layer maps to an output. Traditional feed-forward networks are regularly employed for classification and pattern recognition tasks, but a different structure is most often used for predicting time series data.

There exists a category of neural networks called recurrent neural networks (RNNs). RNNs are a more suitable choice for modeling nonlinear time series data than feed-forward neural networks because they are structured to allow feedback loops in their connection pathways, allowing the system to possess short-term memory. Two commonly used approaches for training RNNs are backpropagation through time (BPTT) and reservoir computing [5]. BPTT requires training all of the weights between nodes of the network. In comparison, reservoir computing frameworks, pioneered independently by Jaeger [6] and Maass [7], have a significantly simpler training process; only the output weights are trained. Note that in the reservoir framework, the network structure of layers is not always employed, and they will not be employed in this work.

2 Project Objectives

Our objective for AMSC 663-664 is to leverage recent advances in machine learning approaches to the modeling and prediction of properties of complex systems to study turbulence in tokamak plasmas. We aim to develop a machine-learning-based tool that can receive state parameters of the plasma and the equations governing the time evolution of the system as inputs and can both accurately and quickly approximate the relevant macroscopic variables necessary to calculate the radial heat flux and other salient physics properties. We have discussed several potential approaches to achieve this objective but have ultimately decided to focus on a reservoir computing framework due to recent promising results [8].

The ideal outcome of the project would be that the trained reservoir algorithm will be able to predict the turbulent dynamics with a faster time to solution than the full-resolution direct numerical simulation provided by the GX turbulence code within a specified tolerance range. However, while the exact details of the plasma's time evolution would be useful knowledge, the most important quantity of interest for our most immediate physics questions is an estimate of Q, the spatiotemporally averaged heat flux.

3 Our Approach

This project will begin by developing code to reproduce the work of Pathak and his collaborators [8], where they have successfully demonstrated a trained reservoir's ability to predict future states of a high-dimensional chaotic system using a framework written in an interpreted language. We employ the same notation. A spectral RK4 solver for the 1-D KS system is implemented implementation in C++ on a local machine. The current reservoir implementation is developed in Python and employs the third-party libraries NumPy and SciPy, which are commonly used in scientific computing applications. Figures are generated in Python using the third-party library Matplotlib. Utilizing the sparse matrix class and methods from SciPy facilitates a fourfold decrease in computation time for the reservoir generation and training when compared to the first Python implementation.

Following Pathak et al., the domain will be discretized into a set of N grid points. The runtime will be separated into two phases, training and prediction, visually represented in Figure 1. In the training phase, the input to the system at each time step will be a vector $\mathbf{u}(\mathrm{t})$ of dimension N, where each element of $\mathbf{u}$ is the solution of the equation at one of the N gridpoints at time t. The reservoir will be a collection of D_r neuron-like units and will be represented in two ways, both as a vector and as a matrix. The vector representation will be as a state vector, $\mathbf{r}(\mathrm{t})$, where each element will represent the state of one artificial neuron at time t. The matrix representation will be as a $D_r \times D_r$ large, sparse adjacency matrix, $\mathbf{A}$, with the an average of 3 nonzero elements per row initialized to random values drawn from the uniform distribution on [0,1] and then scaled by the appropriate factor to fix the largest eigenvalue equal to a value, ρ. (The magnitude of the largest eigenvalue of the reservoir's adjacency matrix representation has been demonstrated to impact the accuracy of the reservoir's predictions. [9]) The input layer $\hat{\mathbf{R}}_{in}$ will be the $D_r \times N$ matrix $\mathbf{W}_{in}$. $\mathbf{W}_{in}$ has its nonzero elements drawn from the uniform distribution on $[-\sigma, \sigma]$. A dense structure for $\mathbf{W}_{in}$ is tested and a sparse, block diagonal structure $\mathbf{W}_{in}$ is tested. The dense structure draws all elements from the uniform distribution on $[-\sigma, \sigma]$. The block-diagonal structure has exactly one nonzero element per row and q nonzero elements per column. In both cases, $D_r = Nq$. Critically, $\mathbf{A}$ and $\mathbf{W}_{in}$ will remain fixed after they are initialized. They are not subject to optimization. During the training phase, the scalar states of the reservoir will update with the equation

$$\mathbf{r}(t + \Delta t) = \tanh[\mathbf{A}\mathbf{r}(t) + \mathbf{W}_{in}\mathbf{u}(t)],$$

and the hyperbolic tangent acts element-wise on the vector argument.

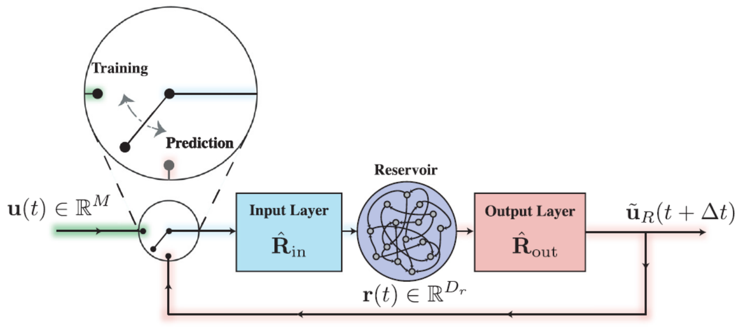
Training
Prediction
Reservoir
$\mathbf{u}(t) \in \mathbb{R}^M$
Input Layer
$\hat{\mathbf{R}}_{\text{in}}$
Output Layer
$\hat{\mathbf{R}}_{\text{out}}$
$\tilde{\mathbf{u}}_R(t + \Delta t)$
$\mathbf{r}(t) \in \mathbb{R}^{D_r}$

Figure 1: Schematic diagram of the reservoir framework we plan to implement for both the 1-D KS equation and for the 5-D gyrokinetic system. from J. Pathak et al. Chaos **28**, 041101 (2018).

The elements of the reservoir-to-output coupling, $\mathbf{W}_{out}$, are parameters that must be tuned to train the reservoir. Additionally, we introduce a hyperparameter T_b to exclude initialization transients from the reservoir training process. This hyperparameter serves as a buffer of time steps between the initialization time and the data assigned for training $\mathbf{W}_{out}$. During the training phase, defined by the times t where $T_b\Delta t \leq t \leq T$, the reservoir output will be trained to approximate the data input. Because the reservoir output, $\mathbf{W}_{out}\mathbf{r}(t)$ is a linear operation, we can train the parameters through linear regression. We introduce a regularization parameter β in the method of Tikhonov-regularized linear regression to reduce model complexity in case of potential overfitting. The optimization problem is then to find the minimum of

$$\sum_{j=T_b}^{T/\Delta t} ||\mathbf{u}(j\Delta t) - \mathbf{W}_{out}\mathbf{r}(j\Delta t)||^2 + \beta||\mathbf{W}_{out}||^2.$$

Explicitly, the matrix multiplication to solve the linear regression problem is

$$\mathbf{W}_{out} = \mathbf{U}\mathbf{R}^T(\mathbf{R}\mathbf{R}^T + \beta I)^{-1}$$

where $\mathbf{U}$ is an $N \times (T/\Delta t - T_b)$ matrix consisting of the time series of u(x,t), $\mathbf{R}$ is an $N \times (T/\Delta t - T_b)$ matrix consisting of the time series of the state vector $\mathbf{r}$(t), $\mathbf{R}^T$ is the transpose of $\mathbf{R}$, and I is the identity matrix.

Once the weights are trained, the algorithm will switch to the prediction phase. As indicated in Figure 1, the system will no longer receive input from the direct numerical solution. Instead, the output of the layer will be fed back into the input layer for the next time step. In the prediction phase, the reservoir's dynamics evolve under the equation

$$\mathbf{r}(t + \Delta t) = \tanh[\mathbf{A}\mathbf{r}(t) + \mathbf{W}_{in}\tilde{\mathbf{u}}_R(t)],$$

where $\tilde{\mathbf{u}}_R(t)$ is the prediction of the reservoir for time t, $\tilde{\mathbf{u}}_R(t) = \mathbf{W}_{out}\mathbf{r}(t)$.

The first iteration of the 1-D KS solver will be a C++ implementation of the fourth-order Runge-Kutta method in Fourier space. If time permits, we will also implement the 1-D KS solver using spectral methods with GX, as GX is a spectral code. This would then allow us, time permitting next semester, to implement a hybrid reservoir system with an embedded, low resolution spectral solver. Such a system would serve as a small-scale test of the eventual use-case, the full 5-D gyrokinetic system. It would also provide another opportunity to compare the results to those that were published using an embedded RK4 solver. It is common for the same machine learning algorithm to yield different success rates when applied to two different problems of the same class, so it would be useful to quantify the impact of embedding a different solver into the hybrid reservoir. The hybrid reservoir scheme functions similarly to the standard scheme with one main distinction: a knowledge-based model is included in the system [10]. This knowledge-based model, in our case a PDE solver running at a lower resolution, serves as an additional input to the input layer of the reservoir and to the output layer of the reservoir. For the hybrid case, the output is a trained linear combination of the knowledge-based model and the reservoir itself.

4 Results

We document our progress toward modeling turbulence in tokamak plasmas in increments. In Section 4.1, we present the validation of the 1-D KS solver with reference to the Kassam and Trefethen solutions [11]. In section 4.2.1, we validate the reservoir code by applying it to the KS system and compare its results to those obtained by Pathak and his collaborators [8]. We then expand on the initial application by configuring the reservoir to directly model the spectral data for the KS system in Section 4.2.2. The particle diffusion coefficient of the KS system is predicted in Section 4.2.3. Finally, the reservoir is applied to zonal flow data derived from a higher-dimensional turbulent system in Sections 4.3 and 4.4.

4.1 KS Solver Validation

Multiple iterations of the KS solver were tested. The code is validated by comparing the results of the C++ spectral solver to the results published in Kassam and Trefethen using the same initial condition. [11] In that work, the authors choose a spatial domain of N = 128 evenly-spaced grid points on the domain $x \in [0, 32\pi]$. The initial condition is

$$u(x, 0) = \cos(x/16)[1 + \sin(x/16)].$$

Initially, we utilized the convolution theorem to evaluate the nonlinear term of the KS equation in Fourier space, and time-stepping utilized the integrating factor method described by Kassam and Trefethen. The function developed to calculate the nonlinear term in Fourier space was compared to the direct calculation of that term in real space for a variety of periodic functions, and the results were identical. Additionally, the first iteration of the code could solve a linear PDE using an integrating factor for time-stepping. However, this initial version of the KS solver was unstable even for small time steps (dt = 0.0001). We discovered that near t = 75, the reality condition for u(x,t) slightly began to break down. The discrete Fourier transform of a real-valued function has complex conjugate symmetry in its spectrum. If we denote $\hat{u}_k$ as the k'th Fourier coefficient of the discrete Fourier transform of u(x,t), then $\hat{u}_k = \hat{u}^{\dagger}_{N-k}$. We discovered that near t = 75, even for time steps dt of different orders of magnitude, the solution in Fourier space began to lose this symmetry property. To rectify this, we restructured the solver to strictly enforce the reality condition on u(x,t) in two ways. First, we calculate only the half-spectrum of u and directly utilize the complex conjugate symmetry. Second, we require the inverse Fourier transform function to return a real-valued vector. Finally, we dealiased the calculation of the Fourier representation by calculating fewer modes. These changes stabilized the solver. In Figure 2, we validate the result of the C++ KS solver by comparing it to the results Kassam and Trefethen obtain [11].

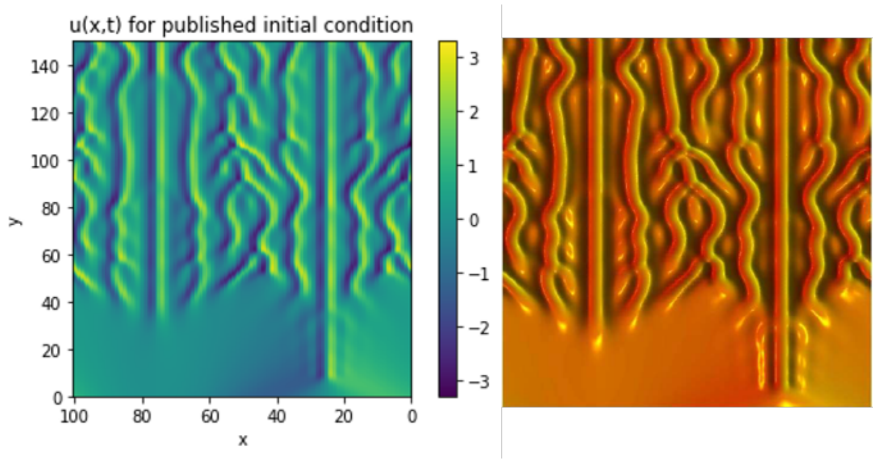

Figure 2: Comparison of the output of the C++ spectral 1-D KS solver (left) to the result published in Fig. 6 of A. Kassam and L. N. Trefethen SIAM J. Sci. Comput., **26**, 4 (2005). The same initial condition was used to generate both solutions.

4.2 Application of Reservoir to KS System

This implementation of the reservoir computing method requires several hyperparameters which must be specified manually. As described in Section 3, these hyperparameters include the number of artificial neuron units in the reservoir D_r, the number of time steps of data available for training the output coupling $T_L = T/\Delta t$, the average degree of the reservoir, the scaling factor for the input coupling weights σ, the spectral radius of the sparse adjacency matrix, and the regularization parameter β. Additionally, T_b is the number of time steps between excluded at the beginning of the training set to mitigate the possibility of initialization transients impacting the training. In the following subsections, q provides a domain-independent metric for the number of neuron-like units in the reservoir D_r, as we define $D_r = Nq$ where N is the number of grid points in the domain of the numerical solver. p specifies the value of the regularization parameter for training the output-coupling matrix: $\beta = 10^p$. In the following figures, r represents the spectral radius of the sparse adjacency matrix $\mathbf{A}$. Finally, s controls the magnitude of the input coupling weights. The nonzero elements of $\mathbf{W}_{in}$ are drawn from the uniform distribution on $[-\sigma, \sigma]$. The hyperparameter σ is determined by $\sigma = 1/s$.

4.2.1 Validation

Figure 3 presents a comparison between the direct numerical solution of the 1-D KS equation and the prediction that the reservoir generates once trained. The reservoir demonstrates performance consistent with the results presented in J. Pathak et al. [8]. For this simulation, the KS solver employs a domain of $N = 64$ spatial grid points and a time step of $\Delta t = 0.25$. The prediction agrees well with the numerical solver for $t = 120$ (480 time steps) after training. In comparison, J. Pathak et al. report valid predictions for 6 Lyapunov times in this system, which corresponds to $t = 60$ [8]. The improved performance presented in Figure 3 can be attributed to differences in the prescribed hyperparameters as later explored in Section 4.4. These numerical results demonstrate that the reservoir code implemented in Python successfully predicts the spatiotemporal dynamics of the KS system for a time that is consistent with a commonly cited publication.

Initial iterations of the reservoir code implemented a dense matrix for $\mathbf{W}_{in}$, where all elements are drawn from the uniform distribution on $[-\sigma, \sigma]$. However, that reservoir structure generated valid predictions for far fewer time steps after the training phase than those presented in Figure 3. Implementing a block-diagonal structure for $\mathbf{W}_{in}$ as described in Section 3 extends the valid time of the solution to the time we currently report. The structure of $\mathbf{W}_{in}$ is critical to the successful prediction of the KS system for these given hyperparameters.

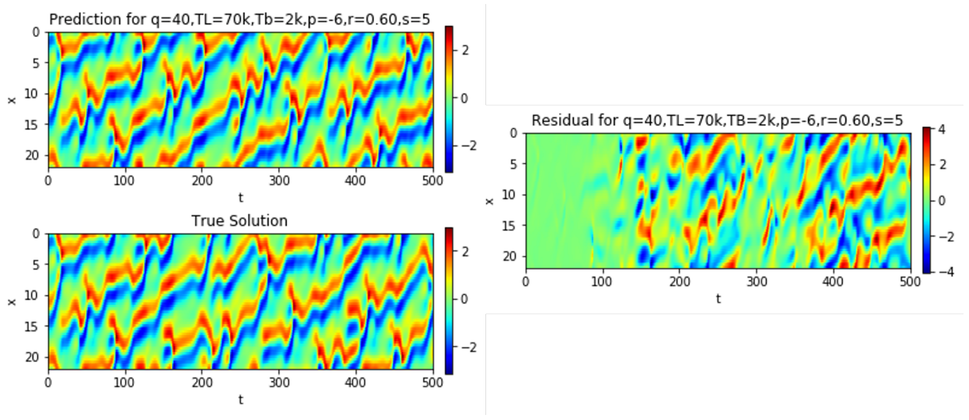

Figure 3: Comparison between the numerical solver and the reservoir prediction for the 1-D KS system. The image positioned at the bottom-left is the numerical solution to the 1-D KS equation with a random initial condition plotted for 500 time units at the end of the training set. The image positioned at the top-left is the reservoir prediction for that time interval with hyperparameters defined as in the beginning of Section 4.2. The third image is the residual, taken as the difference between the prediction and the true solution and normalized by the RMS of the training data.

4.2.2 Application to Spectral Domain

We extend the work of Pathak et al. by applying the reservoir to the spectral domain of the KS system. The numerical solver we implement in C++ solves the KS equation in Fourier space, and due to both the conjugate symmetry of the Fourier transform of real-valued functions and the dealiasing method employed in the spectral solver, the spectral domain contains fewer elements than the real-space domain contains. If it is possible to use the reservoir to reliably predict useful physics quantities using spectral data, then the method may be useful for efficiently predicting physics quantities in higher-dimensional systems. The results of the spectral predictions are presented in Figure 4. The training data time-series consists of a $P \times T_L$ matrix, where each column is the solution of the KS equation evaluated in Fourier space. In every column of the training set, each even-numbered element is the real part of a Fourier mode, and each odd-numbered element is the corresponding imaginary part of that Fourier mode. For this simulation, $P = 44$ and $T_L = 70,000$. Applying the sparse, block-diagonal structure for $\mathbf{W}_{in}$ did not yield valid predictions. The reservoir predictions quickly approached a nearly constant state that is out of the range of the typical values of the solution. The dense $\mathbf{W}_{in}$ structure for the spectral data performed in a manner consistent with the original results shown in Figure 3.

The sparse structure likely fails for the spectral prediction because the input data is no longer governed by local interactions. In Fourier space, the nonlinear term of the KS equation involves a convolution, which is an inherently global operation. The adjacency matrix $\mathbf{A}$ is sparse, with 2944 total elements per row, but only an average of 3 nonzero elements per row. If $\mathbf{W}_{in}$ is also sparse, then the state vector cannot effectively model interactions between distant Fourier harmonics because the reservoir does not facilitate connections between distant portions of the Fourier domain through either the input or the adjacency matrix. This result illustrates the significant impact of the structure of the reservoir input weights on its resulting predictions. Given this result, it may be possible to construct an adjacency matrix that, in addition to being sparse, is informed by knowledge of the underlying physical system.

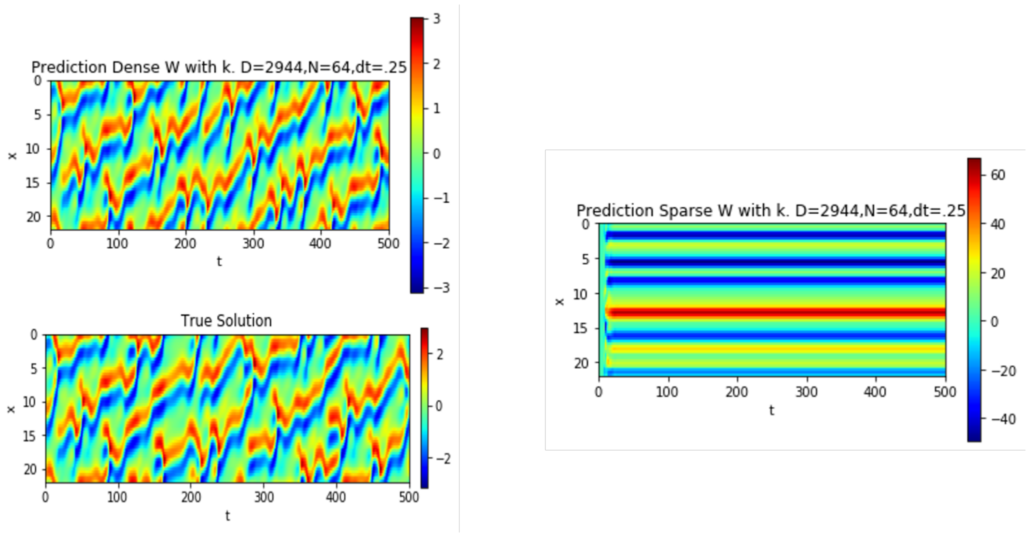

Figure 4: Comparison between predictions using dense $\mathbf{W}_{in}$ and sparse, block-diagonal $\mathbf{W}_{in}$ when applied to a spectral domain. The image positioned at the bottom-left is the numerical solution to the 1-D KS equation with a random initial condition plotted for 500 time units at the end of the training set. The top-left and the rightmost images are the projection of the spectral reservoir predictions onto real-space for visualization. The left prediction uses a dense $\mathbf{W}_{in}$, while the right prediction uses a sparse, block-diagonal $\mathbf{W}_{in}$. In both predictions, there are D units in the reservoir, and N and dt specify the simulation information.

4.2.3 Reservoir Predictions of KS Diffusion Coefficient

While accurately predicting the dynamics of the system would be ideal, some relevant physical quantities that exist in tokamak plasmas are box-averaged and therefore less sensitive to positional offsets in the locations of perturbations like those observed in Figure 3. One of the most critical goals is to model the dependence of the scalar turbulent heat flux on a variety of engineering and physics parameters for tokamak experimental designs. We can begin to assess the probability that the reservoir approach will be successful at this task by performing a similar task on the smaller KS system. The particle diffusion coefficient, D, from the analytic theory in [4] discussed in Section 1.1 can serve as an analogue. We built a diagnostic into the KS solver to track D(t), and the result is plotted on the left of Figure 5. In full gyrokinetic simulations, the mean value of the turbulent heat flux after initial transients resolve is a valuable, but computationally expensive physical quantity to determine. This successful prediction of the mean value of the diffusion coefficient with less than 1% error suggests that it may be possible to use the reservoir to predict the heat flux in the full gyrokinetic system.

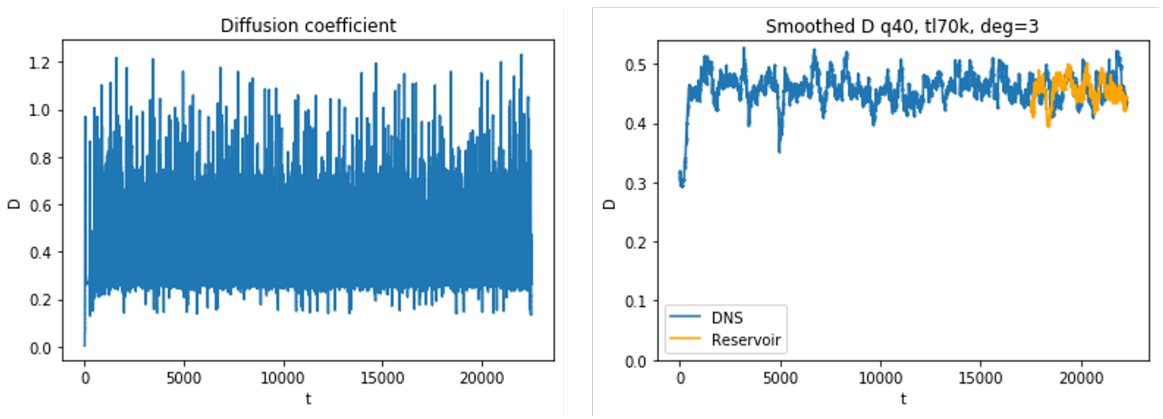

Figure 5: Diffusion coefficient for the KS system. On the left is the diffusion coefficient from the numerical solver alone. On the right is the diffusion coefficient from the numerical solver and the reservoir after both have been smoothed for visualization using a moving average with a time window of 1000 points. The diffusion coefficient mean for the direct numerical solver is 0.45651 for this simulation, and the reservoir predicts a diffusion coefficient mean of 0.45387. The error in the predicted mean is 0.59%.

4.3 Application of Reservoir to Zonal Flow Data from 2-D Turbulent System

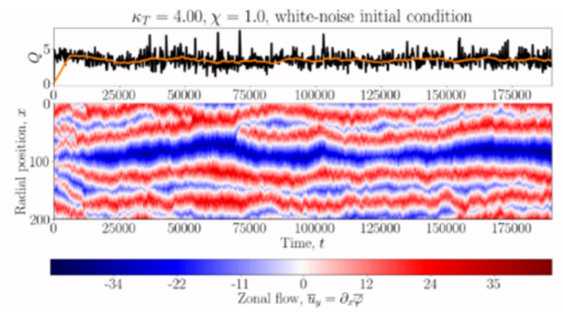

Figure 6: Zonal $\mathbf{E} \times \mathbf{B}$ flow data for a highly collisional, two-dimensional limit of the full gyrokinetic system. Figure 14 of P. Ivanov et al. J. Plas. Phys. **86**, 855860502 (2020).

A recent study by P. Ivanov et al. investigates zonal $\mathbf{E} \times \mathbf{B}$ flows in a 2-D turbulent system [12]. Channels that resemble a less chaotic KS solution arise in the zonal flow data. For this system, the successful prediction of when sharp discontinuities occur in the flow would be an ideal result. Zonal flow data for this system is generated by GX and supplied to the reservoir as an input for training. The domain contains $N = 64$ grid points, and $\Delta t = 1$ for this system. The prediction results are presented in Figure 7. The reservoir successfully approximates the directions of the changes in zonal flow for 100 time steps before diverging beyond the range of the true solution. Due to time constraints, we have not determined the characteristic Lyapunov time for this system, so quantitative comparisons to the KS results are not currently possible. Qualitatively, the Lyapunov time should be longer in this case, as there are fewer high spatial frequency oscillations than in the KS data.

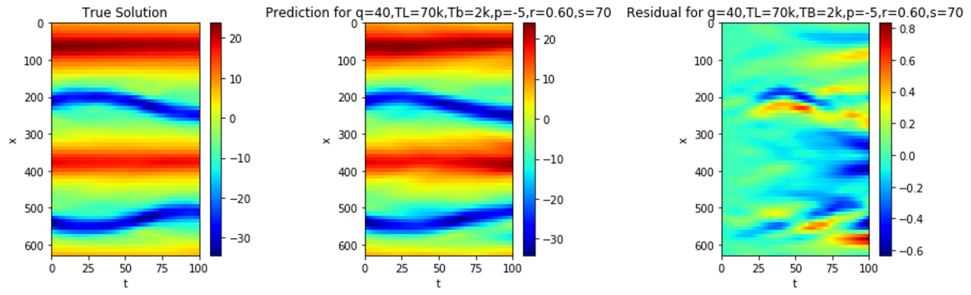

Figure 7: Reservoir predictions of the zonal flow data. The prediction shows strong qualitative agreement with the solution for the first 100 time steps.

4.4 Reservoir Hyperparameter Scan

Following this preliminary result, multiple reservoir realizations are performed to assess the impact of the hyperparameters on the prediction accuracy and stability of the reservoir. The block-diagonal structure is chosen for $\mathbf{W}_{in}$, as the zonal flow data appears to behave primarily with local interactions. The degree of the adjacency matrix $\mathbf{A}$ is set to 3 for all of the figures in this section. The hyperbolic tangent activation function uses the fixed, untrained matrices $\mathbf{W}_{in}$ and $\mathbf{A}$ to advance the reservoir state. Studying the impact of each of these matrices in isolation may yield some additional insight toward developing a prescription for choosing the hyperparameters of the model such that the reservoir prediction remains stable. The range of the elements of $\mathbf{W}_{in}$ is controlled by $\sigma = 1/s$, and the range of the elements of $\mathbf{A}$ is controlled by its spectral radius ρ. Therefore, as s increases, the contribution of $\mathbf{W}_{in}$ to advance the state vector decreases. Similarly, as ρ decreases, the impact of $\mathbf{A}$ on the state vector decreases.

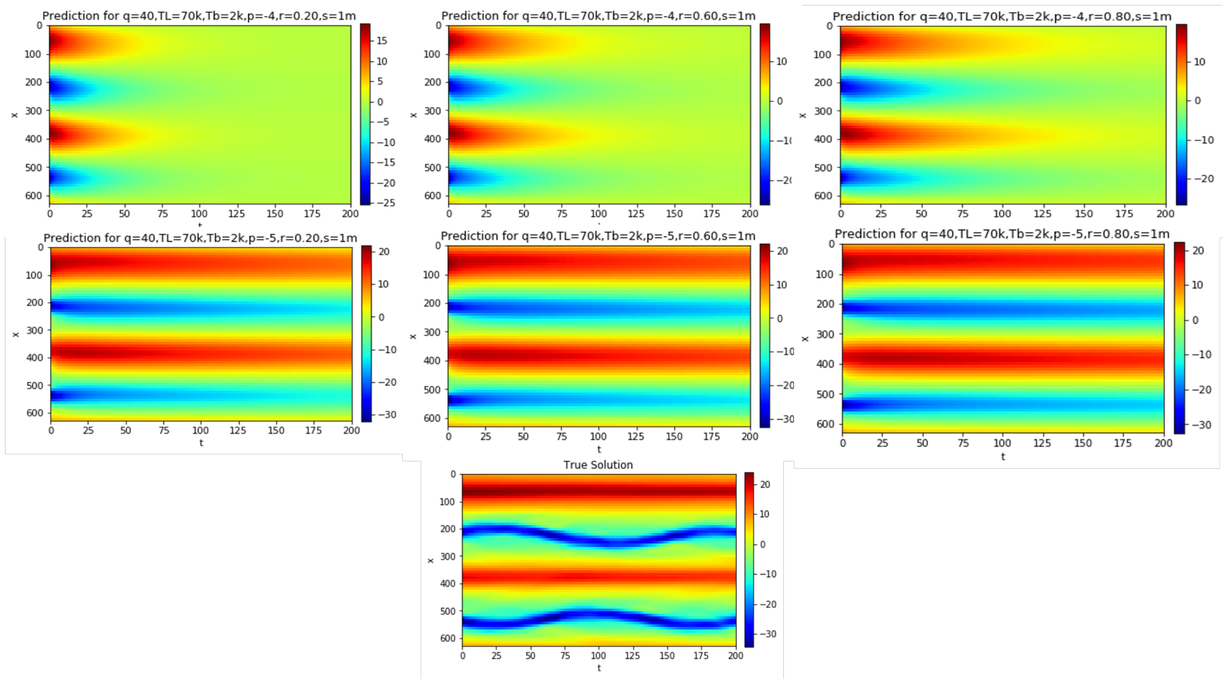

Figure 8: Zonal flow predictions for s = 1,000,000. The top row corresponds to $\beta = 10^{-4}$, and the bottom row corresponds to $\beta = 10^{-5}$. The left column sets $\rho = 0.20$, the middle column sets $\rho = 0.60$, and the right column sets $\rho = 0.20$. Variables are defined as in Section 4.2.1.

For two values of $\beta = 10^{p}$, a scan of three values of the spectral radius with $s = 1,000,000$ is presented in

Figure 8. For the same two values of $\beta = 10^p$ as the previous scan, a scan of three values of s with $\rho = 0.0001$ is presented in Figure 9. As in Section 4.2.1, ρ is labeled as r in the figures.

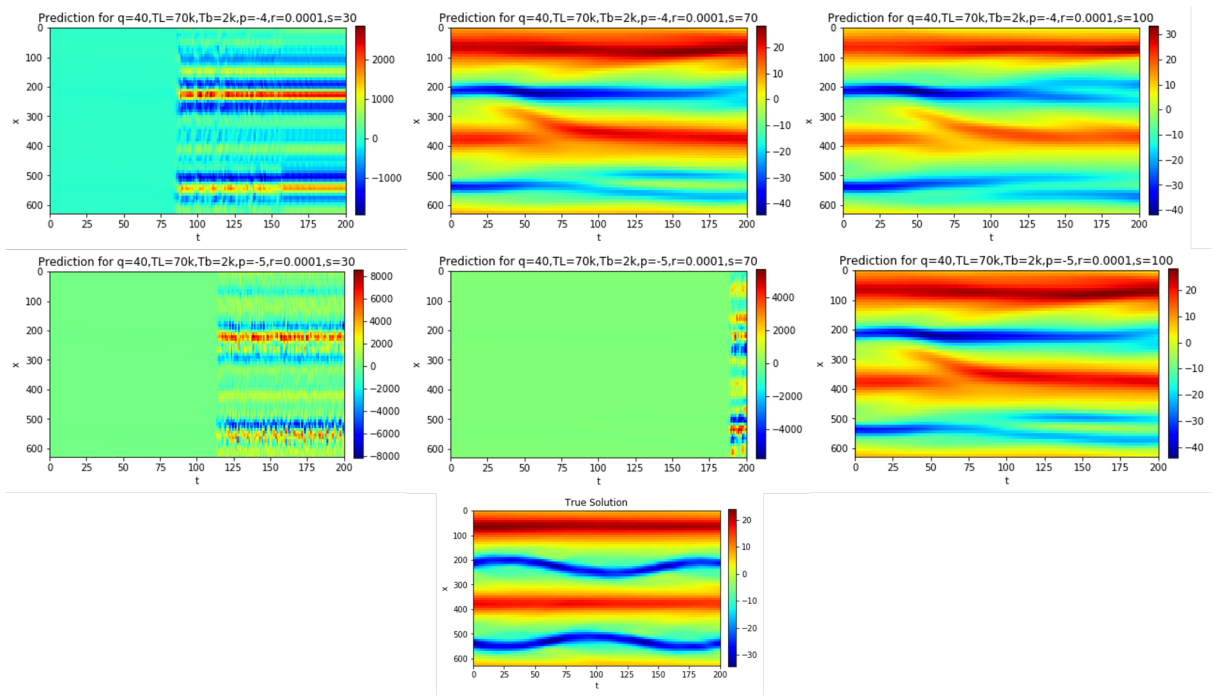

Figure 9: Zonal flow predictions for $\rho = 0.001$. The top row corresponds to $\beta = 10^{-4}$, and the bottom row corresponds to $\beta = 10^{-5}$. The left column sets $s = 30$, the middle column sets $s = 70$, and the right column sets $s = 100$.

The spectral radius scan yields anticipated results. As the spectral radius decreases, the time before the solution approaches to zero decreases. Both scans indicate that the ideal values of σ and ρ must depend on the choice of β. Additional hyperparameters including the training length, buffer length, and average degree of $\mathbf{A}$ must be explored.

5 Future Work

Ultimately, we will build upon this work by studying the reservoir algorithm's ability to model a more complicated PDE with several orders of magnitude of increase in the number of grid points. We will focus on the algorithm's ability to accurately predict the turbulent heat flux, which is an integrated and averaged quantity, as opposed to focusing solely on the fidelity to the direct numerical simulation of the turbulence. The original intent for this project was to develop a CUDA implementation of the reservoir to integrate it into the gyrokinetic turbulence code GX, but due to time constraints, this was not completed. A preliminary draft of the CUDA implementation is complete, but it does not produce predictions that are as accurate as the Python reservoir code for the 1-D KS system. For systems that contain orders of magnitude more data, training the reservoir using the exact algorithm as the Python implementation will be infeasible due to memory constraints, so improvements to the algorithm will be employed.

One of the first tasks following the end of the semester will be to finalize the validation of the CUDA implementation. Once the CUDA implementation is validated, we will return to the KS system and the 2-D system to rigorously continue the hyperparameter study. Then, after developing a prescription for choosing the structure of $\mathbf{W}_{in}$ and the hyperparameters of the model, we will pursue preliminary predictions of the turbulent heat flux of the full gyrokinetic system. The hybrid and parallel reservoir models will be explored and compared to a colleague's Long Short-Term Memory network results.

6 Deliverables

The following items were submitted to the course instructors:

- Final report and presentation
- C++ KS solver, Python reservoir interactive script, and documentation
- Sample input and output files

References

[1] W. Dorland, F. Jenko, M. Kotschenreuther, and B.N. Rogers, Phys. Rev. Lett. **85**, 5579 (2000).

[2] N.R. Mandell, W. Dorland, and M. Landreman, Journal of Plasma Physics **84**, 905840108 (2018).

[3] M. Barnes. Ph.D. Thesis (2008) arxiv:0901.2868

[4] R. E. LaQuey, S. M. Mahajan, P. H. Rutherford, and W. M. Tang Phys. Rev. Lett. **34** pp. 392–394 (1975)

[5] P. Vlachas, J. Pathak, B. Hunt, T. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos. Neural Networks **126** 191-217 (2020).

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

[7] W. Maass, T. Natschläger, and H. Markram. Neural Computation **14**:11, 2531-2560 (2002).

[8] J. Pathak, B.R. Hunt, M. Girvan, Z. Lu, and E. Ott. Phys. Rev. Lett. **120**, 024102 (2018).

[9] J. Jiang and Y. Lai. Phys. Rev. Research **1**, 033056 (2019).

[10] J. Pathak, A. Wikner, R. Fussell, S. Chandra, B.R. Hunt, M. Girvan, and E. Ott. Chaos **28**:4, 041101 (2018).

[11] A. Kassam and L. N. Trefethen SIAM J. Sci. Comput., **26**, 4 (2005).

[12] P. Ivanov, A. A. Scheckochihin, W. Dorland, A. R. Field, and F. I. Parra. J. Plas. Phys. **86**, 855860502 (2020).

[13] Z. Lu, B. Hunt, and E. Ott Chaos **28**, 061104 (2018)